INPUT
7
L. 2800
CORSO PRATICO DI PROGRAMMAZIONE
PER LAVORARE E DIVERTIRSI COL COMPUTER
ROM
RAM
3
8
5
ISTITUTO GEOGRAFICO DE AGOSTINI

CORSO PRATICO DI PROGRAMMAZIONE
PER LAVORARE E DIVERTIRSI COL COMPUTER

Pubblicazione a fascicoli settimanali
edita dall'Istituto Geografico De Agostini

volume I - fascicolo 7

Direttori: Achille Boroli - Adolfo Boroli

Direzione editoriale: Mario Nilo; *settore fascicoli:* Jason Vella

Redazione dell'edizione italiana a cura della:
Logical Studio Communication

Traduzione dall'inglese a cura di: Daniel Quinn

Coordinamento grafico: Otello Geddo

Coordinamento fotografico a cura del Centro Iconografico dell'Istituto Geografico De Agostini

Direzione:
Novara (28100), via Giovanni da Verrazano 15 - tel. (0321) 471201-5

Redazione:
Milano (20149), via Mosè Bianchi 6 - tel. (02) 4694451

Amministrazione, abbonamenti e servizio arretrati:
Istituto Geografico De Agostini - Novara (28100), via Giovanni da Verrazano 15 - tel. (0321) 471201-5.

Copertine e risguardi per i volumi dell'opera saranno messi in vendita a L. 6000 la copia (L. 7500 all'estero).

Le copie arretrate saranno disponibili per un anno dal completamento dell'opera e potranno essere prenotate nelle edicole o direttamente presso l'Editore. Per i fascicoli arretrati, trascorse 12 settimane dalla loro pubblicazione, è applicato un sovrapprezzo di L. 400 sul prezzo di copertina in vigore al momento dell'evasione dell'ordine. Spedizione contro rimessa di pagamento anticipato; non vengono effettuate spedizioni contrassegno.

L'Editore si riserva la facoltà di modificare il prezzo nel corso della pubblicazione, se costretto da mutate condizioni di mercato.

Registrato presso il Tribunale di Novara n. 11 in data 19-5-1984.

Direttore responsabile: Emilio Bucciotti

Spedizione in abbonamento postale Gruppo II/70 (Autorizzazione della Direzione provinciale delle PP.TT. di Novara).

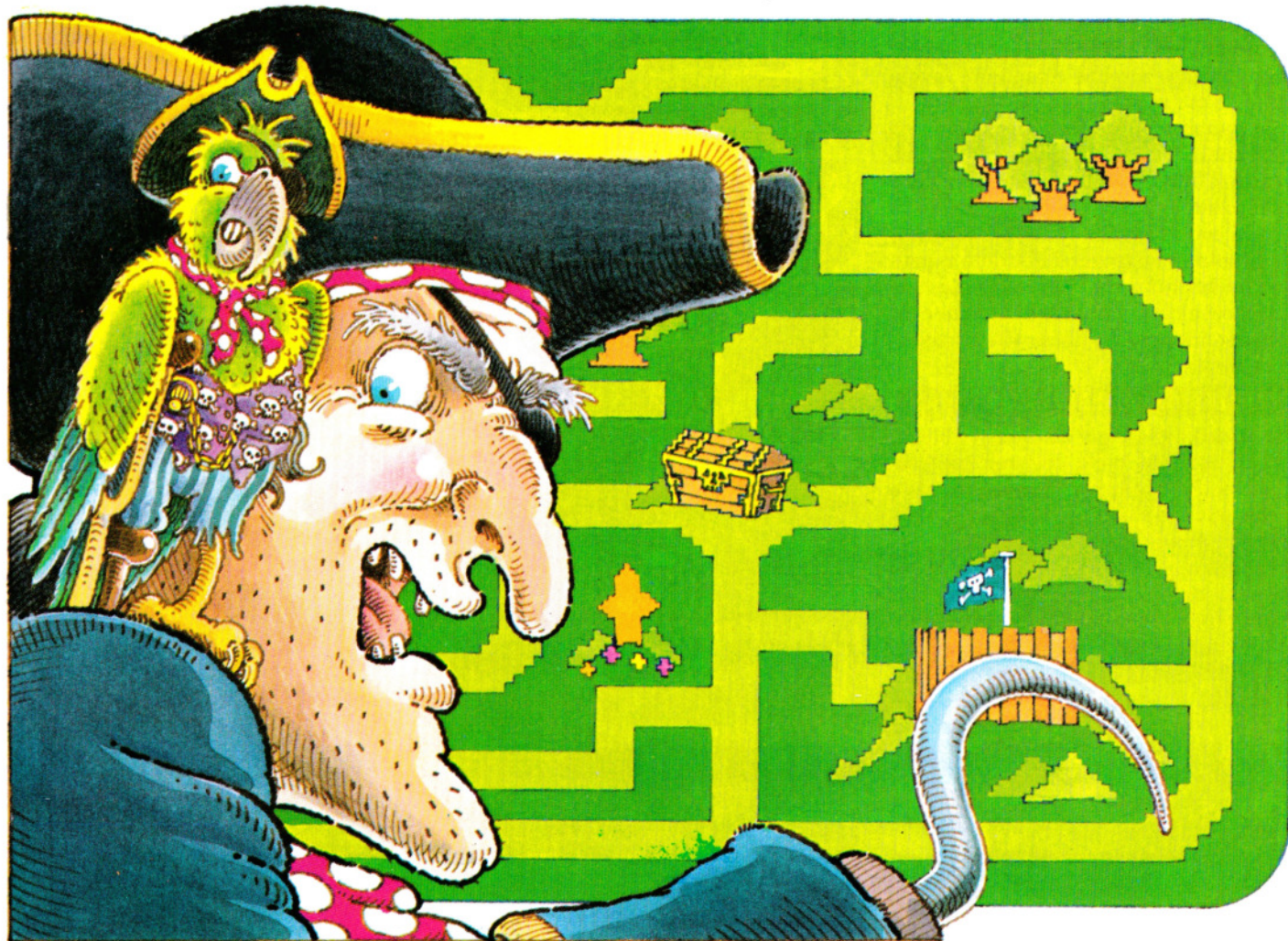
Distribuzione A. & G. Marco - Milano, via Fortezza 27 - tel. (02) 2526.

Pubblicazione a fascicoli settimanali. Esce il martedì.

Stampato in Italia - I.G.D.A. Officine Grafiche, Novara - 048412.

Referenze dei disegni e delle fotografie:
Copertina: Jon Couzins. Pagg. 193, 195, 196, 199 Paddy Mounter. Pagg. 198, 200 Ray Duns. Pagg. 201, 202, 203, 204, 205, 206, 207 Tudor Art Studios. Pag. 208 Jon Couzins. Pag. 210 Tony Roberts. Pagg. 210, 211, 213, 214, 215 Bernard Fallon. Pag. 220 Nick Mijnheer. Pag. 222 Howard Kingsnorth. Joystick gentilmente messi a disposizione da Sonic Foto e Micro Center & Lion House, entrambi situati in Tottenham Court Road, London W1.

PIÙ LIVELLI DI DIFFICOLTÀ

- DISEGNARE UN LABIRINTO CASUALE
- DUE MODI PER RENDERE IL GIOCO PIÙ DIFFICILE
- COME MUOVERE IL PERSONAGGIO
- IL CONTEGGIO DEI PUNTI

Questo gioco di labirinto prevede due livelli di difficoltà e ogni volta presenta un labirinto nuovo. L'obiettivo è raggiungere il tesoro nel minor tempo possibile.

Spesso nei giochi si ha la possibilità di scegliere tra diversi livelli di difficoltà, in modo da far divertire sia i principianti che gli esperti senza essere troppo difficile per gli uni o troppo facile per gli altri.

Esistono diverse maniere per introdurre vari livelli di difficoltà, che dipendono dalla particolare natura del gioco. Per esempio, si può cambiare il numero dei nemici, introdurre pause o limiti di tempo più o meno impegnativi e così via.

In questa lezione, vediamo come incorporare più livelli di difficoltà in un gioco di labirinto. Come vedremo, esistono due strade per raggiungere lo scopo, secondo il computer impiegato. Il gioco del labirinto non consiste nel cercare una via d'uscita, bensì nel guidare un personaggio alla ricerca di un tesoro, senza oltrepassare un tempo massimo prefissato. Si noti che non c'è una versione adatta allo ZX81.

Per facilitare o complicare lo svolgimento, si possono usare due metodi: si può intervenire sulla complessità del labirinto stesso o sulla quantità di tempo a disposizione. La ragione che costringe ad adottare sistemi diversi secondo il computer proviene dal differente criterio impiegato nella creazione dei labirinti.

Il nostro esempio serve anche per dimostrare che il progetto di un gioco a più livelli di difficoltà comporta una precisa scelta preliminare sul metodo da seguire.

LE VITE A DISPOSIZIONE

Quando scade il tempo massimo previsto per cercare il tesoro, è giusto imporre una qualche penalizzazione. Si possono far perdere dei punti al giocatore, ma in genere si usa sottrargli una vita. In questo gioco vengono date al giocatore tre vite, per cui, se per tre volte non riesce a trovare il tesoro nel tempo massimo, il gioco termina.

LABIRINTI CASUALI

Il gioco presentato ricorre all'impiego di una subroutine per la generazione casuale del labirinto: un modulo di programma già di per sé interessante, in quanto disegna ogni volta un labirinto diverso, senza doverne progettare un'intera serie prima di giocare. Si ricorderà che a pagina 68 è stato spiegato come progettare un labirinto usando frasi DATA da incorporare nei programmi: possiamo dunque immaginare quanto sarebbe complicato inventarne un'intera serie.

La creazione di labirinti casuali è molto più facile di quanto si possa pensare. Una soluzione ovvia potrebbe essere stampare sullo schermo a caso un certo numero di blocchi con la grafica ROM, ad esempio. Però, potremmo trovarci di fronte a qualcosa del tutto diverso da un labirinto, non essendoci alcuna garanzia di produrre un percorso interno che abbia effettivamente un'uscita.

COME DISEGNARE LABIRINTI CASUALI

Il metodo migliore per disegnare labirinti casuali consiste nello scrivere un programma che tracci un percorso casuale e costruisca su di esso il labirinto. Il programma è concepito in modo che il percorso non torni mai su se stesso e resti all'interno di una cornice disegnata sullo schermo. Se il percorso casuale non può procedere (perché è finito o in un angolo, o in un vicolo cieco, oppure si è richiuso su se stesso), il computer torna sui propri passi e cerca, nella zona circostante, una via d'uscita. Quando la trova, viene creata una diramazione al percorso, che continua fino al successivo impedimento e il processo si ripete.

Il computer prosegue con le diramazioni, fino a riempire lo spazio nella cornice e a ritornare al punto di partenza. Conclusa la generazione del labirinto, esso contiene un solo percorso, dall'aspetto abbastanza scontato, poiché le diramazioni non sono mai molto complicate. Inoltre, dal labirinto si può facilmente uscire applicando la regola del "tenere la destra", ossia seguendo sempre il lato destro della parete del labirinto (o, per analogia, il sinistro). Creando delle "isole" nel labirinto, si possono confondere le idee a chi cerchi una soluzione così banale. Così, a disegno finito, il programma sparge nel labirinto alcuni blocchi, che fanno sembrare il percorso più complesso di quanto non sia realmente. Trascritto il programma, lo si registri su nastro: la prossima lezione spiegherà come aggiungere anche qualche effetto sonoro.

S

La versione per lo Spectrum inizia con la preparazione di UDG, inizializzando le variabili e predisponendo globalmente il gioco. Senza eseguirla, per ora si trascrivano queste linee:

```
10 FOR m=Ø TO 23: READ a: POKE USR "a"
   +n,a: NEXT n
20 LET hs=Ø
30 INPUT "Livello (da 1 a 6)□";ta
40 LET ta=11ØØ-1ØØ*ta
50 BORDER 1: PAPER 1: INK Ø: CLS: INK 7
60 LET s=Ø: LET lives=3
70 PRINT BRIGHT 1; PAPER 6; INK 2;"□
   PUNTI□□□□□□□□
   PUNTEGGIO MASSIMO□□□□□□□□□"
490 DATA 24,24,6Ø,82,82,24,36,36,127,65,93,
   85,81,95,64,127,24,24,255,255,24,24,24,
   24
```

La linea 1Ø definisce, leggendo le DATA alla linea 49Ø, gli UDG necessari: un personaggio, qualche tesoro e una croce. La linea 2Ø fissa il punteggio massimo (hs) a zero.

Poi al giocatore viene chiesto di scegliere un livello di difficoltà da 1 a 6: più basso è il numero, più facile è il gioco. Si noterà che, alla linea 4Ø, i numeri più bassi stabiliscono tempi massimi più lunghi e viceversa.

La linea 5Ø seleziona i colori del disegno, la 6Ø inizializza il punteggio a Ø e le vite a 3. Infine, la 7Ø visualizza sullo schermo le parole PUNTEGGIO e PUNTEGGIO MASSIMO, adeguatamente spaziate.

IL DISEGNO DEL LABIRINTO

Si scrivano queste linee:

```
80 FOR n=22561 TO 22589: POKE n,16:
   POKE n+64Ø,16: NEXT n
90 FOR n=1 TO 21: POKE 22528+n*32,16:
   POKE 22558+n*32,16:POKE 22559+n*
   32,9:NEXT n
100 LET b=22593: LET a=b
110 DIM a(4): LET a(1)=-1: LET a(2)=-
   32: LET a(3)=1: LET a(4)=32
120 POKE a,56
130 LET j=INT (RND*4)+1: LET g=j
140 LET b=a+a(j)*2: IF PEEK b=8 THEN
   POKE b,j: POKE a+a(j),56: LET a=b:
   GOTO 13Ø
150 LET j=j+1: IF j=5 THEN LET j=1
160 IF j<>g THEN GOTO 14Ø
170 LET j=PEEK a: POKE a,56: IF j<5
   THEN LET a=a-a(j)*2: GOTO 13Ø
180 POKE 22625,56
190 FOR n=1 TO 2Ø
200 LET k=22528+64*(INT (RND*9)+2)
   +INT (RND*29)+1
210 POKE k,56: NEXT n
```

Il contorno del labirinto viene allestito dalle linee 8Ø e 9Ø: con POKE si deposita il valore 16 nell'area di memoria riservata agli attributi della memoria, cosicché viene assegnato il colore rosso a tutto il contorno. Le linee da 1ØØ a 18Ø disegnano il labirinto: se adesso premessimo il tasto BREAK e cancellassimo lo schermo, perderemmo irrimediabilmente il labirinto, in quanto esso è conservato unicamente nella memoria destinata agli attributi.

Le linee da 19Ø a 21Ø completano il labirinto disponendovi a caso 2Ø quadrati. Se questi capitano su una parete, il quadrato diviene parte del percorso.

LA CREAZIONE DEL GIOCO

La seguente sezione di programma riguarda il gioco vero e proprio:

```
220 LET x=15: LET y=1Ø
230 LET tx=INT (RND*15)*2+1
240 LET ty=INT (RND*1Ø)*2+2
250 PRINT BRIGHT 1; PAPER 2;AT Ø,7;s;AT
   Ø,24;hs
260 POKE 23672,Ø: POKE 23673,Ø
270 PRINT FLASH 1; PAPER 3; INK 6;AT
   ty,tx;CHR$ 145
280 PRINT INK 2; PAPER 7;AT y,x;CHR$ 144
290 IF PEEK 23672+256*PEEK 23673>ta
   THEN GOTO 39Ø
300 IF INKEY$="" THEN GOTO 29Ø
310 LET a$=INKEY$: LET sx=x: LET sy
   =y
320 IF a$="z" AND ATTR (y,x-1)>=
   56 THEN LET x=x-1
330 IF a$="x" AND ATTR (y,x+1)>=
   56 THEN LET x=x+1
340 IF a$="k" AND ATTR (y-1,x)>=
   56 THEN LET y=y-1
350 IF a$="m" AND ATTR (y+1,x)>=
   56 THEN LET y=y+1
360 PRINT PAPER 7; INK 2;AT sy,sx; "□";AT
   y,x;CHR$ 144
370 IF ty=y AND tx=x THEN GOTO 47Ø
380 GOTO 29Ø
```

A ogni inizio di gioco, la linea 22Ø fissa la posizione di partenza del personaggio nel punto 15,1Ø.

Il tesoro viene collocato a caso, nel labirinto, dalle linee 23Ø e 24Ø. La linea 27Ø lo visualizza sullo schermo. Le linee 23Ø e 24Ø restringono l'arco di numeri casuali in modo da garantire che il tesoro capiti sul percorso. La linea 25Ø visualizza i valori del PUNTEGGIO e del PUNTEGGIO MASSIMO, inizialmente a Ø e la 26Ø azzera l'orologio, intervenendo con POKE su due locazioni di memoria come esposto a pagina 1Ø1. La linea 29Ø controlla che non venga superato il tempo massimo e, se ciò accade, salta alla 39Ø.

La linea 28Ø visualizza il personaggio.

Nelle linee 270 e 280, per visualizzare gli UDG sullo schermo, vengono usati CHR$144 e CHR$145, un metodo ritenuto migliore a quello che impiega le lettere esposto a pagina 44.

Le restanti linee, dalla 300 alla 380, riguardano il movimento del personaggio attraverso il labirinto. Le linee da 320 a 350 controllano la pressione dei tasti e verificano che il movimento sia lecito. Quanto all'uso di ATTR, per quest'ultimo controllo si rimanda alla spiegazione data a pagina 69, per il gioco del labirinto semplificato. La linea 260 cancella il personaggio dalla posizione precedente, per poi visualizzarlo nella nuova. La linea 370 verifica se si è raggiunto il tesoro: in caso affermativo, aumenta il punteggio prima di offrire un nuovo tesoro da scovare.

SI MUORE SOLO TRE VOLTE

Trascritta la parte finale del gioco, qui sotto, il programma può essere *lanciato*.

```
390 PRINT FLASH 1; PAPER 0; INK 5; AT
    y,x;CHR$ 146
400 LET vite=vite-1: FOR f=1 TO 200:
    NEXT f: IF vite>0 THEN GOTO 260
420 IF s>hs THEN LET hs=s
420 PRINT BRIGHT 1; PAPER 2; AT 0, 24;hs
430 PRINT FLASH 1;AT 10,1;"□PREMERE
    UN TASTO QUALSIASI PER GIOCARE
    ANCORA□"
440 IF INKEY$<>"" THEN GOTO 440
450 IF INKEY$="" THEN GOTO 450
460 GOTO 30
470 LET s=s+ta-PEEK 23672
    +256*PEEK 23673: GOTO 230
```

La linea 390 visualizza una croce lampeggiante (CHR$146) quando il giocatore perde una vita, sottraendola alle vite rimaste (linea 400). Dopo una pausa, il gioco ritorna alla linea 260 che azzera l'orologio per la prossima caccia al tesoro, se si è ancora "vivi". Altrimenti, la linea 410 confronta il punteggio realizzato con quello massimo: se questo è stato superato, viene aggiornato, dopodiché la linea 420 visualizza il massimo raggiunto.

La linea 440 è necessaria se il giocatore sta ancora premendo uno dei tasti di movimento, impedendo così alla 450 di ricevere il segnale per un'altra partita. Infine, la linea 470 calcola il punteggio se la posizione del personaggio e del tesoro coincidono, fatto verificato dalla linea 370.

Impartito il RUN, viene richiesto il livello di difficoltà. La risposta può essere 1, 2, 3 o 4 e corrisponde, rispettivamente, a un tempo di gioco di 20, 16, 12 e 8 secondi.

Scopo del gioco è raggiungere il tesoro (un asterisco collocato casualmente) nel più breve tempo possibile. Il personaggio è un carattere pigreco, guidato al bersaglio dai tasti [Z], [X], [P], e [L]:

```
50 POKE53280,6:INPUT"■◣
   LIVELLO (DA 1 A 4)■";A:IFA<1ORA>
   4THEN50
60 LE=5-A:LE=LE*4+4
100 PRINT"■■■":A=1186:
    POKE 650,255
105 FORZ=0TO39:POKE1104+Z,102:POKE
    1984+Z,102:POKE55376+Z,1:POKE
    56256+Z,1:NEXT
110 A(0)=-1:A(1)=-40:A(2)=+1:A(3)
    =40:FOR F=1 TO 21
150 PRINT"■▒■π□□□□□□□□□
    □□□□□□□□□□□□□□□□□□
    □□□□□□□□□□□■■▒";:
    NEXT F:POKEA,4
220 J=INT(RND(1)*4):G=J:POKE54272+A,
    7
230 B=A+A(J)*2:IFPEEK(B)=160THEN
    POKEB,J:POKE54272+A,6:POKEA+A(J),
    32:A=B:GOTO220
240 J=(J+1)*-(J<3):IFJ<>GTHEN230
250 J=PEEK(A):POKEA,32:IFJ<4THENA=
    A-A(J)*2:GOTO 220
1000 LV=3:FORZ=1TO60:X=INT(RND(1)*
     40+1)+INT(RND(1)*9)*80
1002 IFPEEK(1184+X)=160ANDPEEK(1224
     +X)=160THENPOKE1184+X,32
1004 NEXT Z
```

```
1006 X=RND(1)*720:IFPEEK(1223+X)<>
   32THEN1006
1008 OL=1223+X:POKEOL,42
1010 TI$="000000":POKEA,94:J=3
1013 PRINT"■";:FORZ=1TO62:PRINT"□";:
   NEXT:PRINT"▒PUNT. MASSIMO:■"HS
1015 PRINT"■▒LV:■"LV"□□□□▒
   TEMPO:■"TI$"□□□□□□▒PUNTI:
   ■"SC:IFVAL(TI$)>=LE THEN2000
1016 GETZ$:IFZ$=""THEN1015
1017 IF Z$="Z"THENJ=0
1018 IF Z$="X"THENJ=2
1019 IF Z$="P"THENJ=1
1020 IF Z$="L"THENJ=3
1021 B=A+A(J):IFPEEK(B)<>
   102 AND PEEK(B)<>160 THEN 1040
1030 GOTO 1015
1040 IFPEEK(B)=42THEN3000
1050 POKEB,94:POKEA,32:A=B:GOTO 1015
2000 LV=LV-1:FORZ=155TO0STEP-1:
   POKEA,RND(1)*6+109
2030 NEXT:POKEA,94:IFLV>0THEN1010
2005 PRINT"■□□□■F1□◥ -□
   NUOVO LABIRINTO□□□□□□□□□□□
   □▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
   ▌□□□■F7□◥ -□□INIZIO"
2006 SC=0:LV=3:GETK$:IFK$="    "THEN
   50
2007 IFK$="█|"THEN1010
2010 GOTO2006
3000 SC=SC+50-VAL(TI$):POKEOL,32:IF
   SC>HSTHENHS=SC
3010 GOTO1006
```

Sul Vic 20, occorre sostituire le seguenti linee a quelle di pari numero nel programma per il Commodore 64:

```
50 POKE 36879,110:INPUT"♥◥LIVELLO
   (DA 1 A 4)■";A:IFA<1 OR A>4THEN
   50
100 PRINT "♥■■":A=7770:POKE
   650,128
105 FORZ=0TO21:POKE7724+Z,102:POKE
   8142+Z,102:POKE38480+Z,1:POKE
   38862+Z,1:NEXT
110 A(0)=-1:A(1)=-22:A(2)=1:A(3)=
   22:FORF=1TO18
150 PRINT "■▒■π□□□□□□□□□
   □□□□□□□□□□□□□■■▒";:
   NEXT F:POKEA,4
220 J=INT(RND(1)*4):G=J:POKE30720+
   A,7
230 B=A+A(J)*2:IF PEEK(B)=
   160 THEN POKEB,J:POKE30720<A,6:
   POKEA+A(J),32:A=B:GOTO220
1000 LV=3:FORZ=1TO60:X=INT(RND
   (1)*22)+1+INT(RND(1)*8)*44
1002 IFPEEK(7724+X)=160ANDPEEK(7746
   +X)=160THENPOKE7724+X,32
1006 X=RND(1)*396:IFPEEK(7724+X)<>
   32THEN1006
1008 OL=7724+X:POKEOL,42
1013 PRINT"■";:FORZ=1TO33:PRINT"□";:
   NEXT:PRINT"▒HS:";HS
1015 PRINT "■▒LV:■"LV"□□□□□
   ▒TEMPO:■"TI$"▒PT:■"PT:IFVAL
   (TI$)>=LE THEN 2000
2005 PRINT"■□■F1◥
   -NUOVO LABIRINTO□■F7◥ -INIZIO
   □"
```

Ambedue i programmi per i Commodore iniziano preparando il colori dello schermo e chiedendo il livello di gioco desiderato (linea 50). La routine che crea il labirinto va dalla linea 100 alla 250. Dopo aver disegnato una cornice a scacchi, il programma ne ricopre l'area interna con quadretti gialli, usando la stringa contenuta nella linea 150: i limiti di questa zona sono fissati dalla linea 110.

Lo schema casuale del labirinto è prodotto dalla linea 220, in cui J può assumere un qualsiasi valore da 0 a 3. Questo viene poi convertito nei caratteri @, A, B e C, che rappresentano le direzioni sinistra, alto, destra e basso. Durante la creazione del labirinto, dopo il RUN, questi caratteri lampeggiano brevemente sullo schermo. Ogni cambiamento di direzione lascia dietro di sé uno spazio vuoto, nello stesso colore dello sfondo, che viene a costituire parte del percorso del labirinto. Il più del lavoro viene svolto dalla linea 230.

In effetti, i caratteri @, A, B e C vengono depositati in memoria durante la costruzione del labirinto e non compaiono sullo schermo, eccetto che momentaneamente, per essere subito dopo mascherati dallo spostamento del cursore. Se quest'ultimo non si è potuto muovere nella prima direzione casuale, ne prova un'altra, tornando sui propri passi soltanto se non esistono più caselle gialle. A ogni passo, il cursore tenta di spostarsi ovunque, tranne che indietro ed ecco perché le aree di spazi gialli, originariamente evitate, sono alla fine inglobate nel labirinto.

L'ultima parte della linea 240 controlla se il cursore ha raggiunto la posizione di partenza e, in caso contrario, il programma passa alla linea successiva, che stabilisce una nuova direzione e fa ripartire la sezione per la costruzione del labirinto.

In seguito, le linee da 1000 a 1040 collocano a caso altri spazi vuoti lungo le pareti. La linea 1006 sceglie, ancora a caso, la posizione dell'asterisco (il tesoro), che viene poi visualizzato con la POKE della linea seguente. Il programma azzera quindi il timer incorporato, cancella la parte di schermo dove poi compariranno le scritte e visualizza: il punteggio massimo (variabile HS), le vite a disposizione (LV-1), il tempo trascorso dall'azzeramento del timer (TI) e il punteggio corrente (SC).

La parte finale della linea 1015 controlla se il tempo è esaurito.

Le altre cinque linee del programma verificano la pressione dei tasti, utilizzando il valore corrispondente alla nuova posi-

zione nella prima parte della linea 1Ø21. Le successive PEEK controllano che la direzione sia giusta e che non si sia andati contro una parete. La routine alle linee 1Ø4Ø e 1Ø5Ø verifica se si è raggiunto l'asterisco e in caso contrario fa comparire il personaggio nella sua nuova posizione.

La posizione viene registrata (A = B) e il programma viene rimandato, con una GOTO, alla linea 1Ø15, che aggiorna il punteggio e controlla il tempo trascorso.

Se questo è scaduto, una GOTO manda il programma alle linee 2ØØØ, dove il numero totale di vite (LV) è ridotto di 1 e il personaggio viene fatto "tremolare", grazie a una rapida successione di 155 caratteri. A questo punto, se si hanno ancora vite a disposizione, la linea 2ØØ3 ripristina il personaggio, riportando poi il programma alla linea 1Ø1Ø. Se invece le vite sono esaurite, vengono visualizzate le informazioni per un nuovo gioco e tutto riparte da capo. La linea 2ØØ6 predispone nuovamente le variabili relative al punteggio e alle vite, controllando anche se viene premuto il tasto funzione [F1] e, nel caso, creando un nuovo labirinto. La linea 2ØØ7 risponde invece alla pressione del tasto funzione [F7], riavviando il gioco.

Se durante il gioco si riesce a raggiungere l'asterisco entro il tempo massimo, le linee dalla 3ØØØ alla 3Ø1Ø aggiornano il punteggio, cancellando l'asterisco e totalizzando il nuovo punteggio massimo.

I computer dell'Acorn utilizzano un cambio di MODE per ottenere due livelli di difficoltà: siccome il MODE Ø ha un numero di colonne doppio rispetto al MODE 1, la complessità del percorso può essere comodamente raddoppiata. Pertanto, al MODE 1 corrisponde un labirinto più facile, mentre al MODE Ø ne corrisponde uno più difficile.

Ecco la parte principale del programma. Più oltre vengono definite le procedure i cui nomi, comunque, sono già indicativi circa le funzioni svolte.

Le variabili usate in questa fase sono: D, il livello di difficoltà, X e Y, la posizione di partenza del personaggio, M, il numero di vite e SC, il punteggio realizzato.

```
10 PROCpreparaz
20 MODE(1—D)
30 PROCcostruzione
40 PROCvisione
50 TIME=Ø
60 X=32/(D+1):Y=1023—4*32
70 VDU5
80 REPEAT
90 PROCgioco
100 UNTIL M<Ø
110 PROCfine
120 MODE1
130 PRINTTAB(15,15)"PUNTI□";SC
140 PRINTTAB(15,20)"PREMI RETURN"
150 PRINTTAB(15,21)"PER UN ALTRO
   GIOCO"
160 *FX21
170 IF INKEY$(—74) THEN RUN
180 GOTO 170
190 END
```

L'uso delle procedure rende maggiormente comprensibile l'intera struttura del gioco. Purtroppo, non è consentito cambiare il MODE all'interno di una procedura: occorre quindi che la linea 2Ø faccia parte della sezione principale del programma. Sarebbe stato più ordinato inserire le linee dalla 12Ø alla 18Ø nella PROCfine, ma anche stavolta il cambio di MODE, prima della visualizzazione delle istruzioni, costringe a comportarsi diversamente. La prima procedura, PROCpreparaz predispone tutte le variabili e definisce gli UDG:

```
200 DEFPROCpreparaz
210 *TV254,1
220 DIM S(3)
230 PRINT"DIFFICOLTA: Ø—FACILE 1
   —MENO FACILE?"
240 D=GET—48
250 IF D<>1 AND D <>Ø THEN 240
260 M=2:SC=Ø:MT=3000
270 *FX11,10
280 *FX12,10
290 VDU23,255,255,255,255,255,255,255,
   255,255
300 VDU23,224,255,255,255,255,255,255,
   255,255
310 VDU23,225,24,24,60,90,90,24,36,36
320 VDU23,226,0,255,129,189,165,173,161,
   191
330 VDU23,227,24,24,255,255,24,24,24,24
340 ENDPROC
```

Le linee 23Ø-25Ø si occupano del livello di difficoltà. La linea 24Ø accetta ed interpreta il codice ASCII del numero digitato: poiché il valore ASCII di Ø è 48 e il valore di 1 è 49, per riottenere Ø e 1 si sottrae 48. Digitando altri numeri non validi, si è rimandati indietro dalla linea 25Ø. La linea 21Ø fa abbassare l'immagine video di una linea e la 22Ø dimensiona la matrice S, che serve per garantire che esista sempre una parete di separazione tra due percorsi affiancati. La 26Ø assegna 3 vite (da Ø a 2), un punteggio zero e un tempo massimo di 3Ø secondi a ogni ricerca del tesoro. Sull'Electron questo tempo può risultare troppo breve e va cambiato in MT=5ØØØ. Le chiamate *FX, alle linee 27Ø e 28Ø, velocizzano la ripetizione automatica dei tasti e le ultime cinque linee definiscono gli UDG.

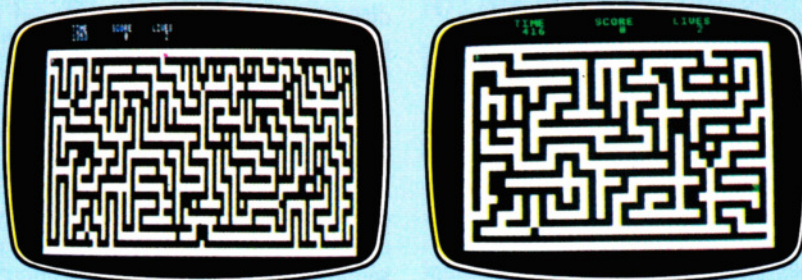
Ambedue i caratteri 224 e 255 sono caselle piene, il primo adoperato per la parte principale del labirinto, il secondo per la parete di contorno. Anche se sullo schermo appaiono identici, il diverso codice permette al computer di far sì che il percorso non attraversi il contorno.

Gli altri caratteri sono per il personaggio (225), per il tesoro (226) e per una croce (227), che compare quando si perde una vita.

IL DISEGNO DEL LABIRINTO

La seconda procedura crea il labirinto nella memoria del computer:

```
350 DEFPROCcostruzione
360 VDU23;8202;0;0;0;
370 A=&7000+41+40*D
380 S(0)=—1:S(1)=—40—D*40:S(2)=1:
   S(3)=40+D*40
390 FOR Z=0 TO (38)+D*40):?(&7000+Z)
   =224:?(&7000+Z+960+D*960)=
   224:NEXT
400 FOR F=1 TO 23:?(F*(40+D*40)+
   &7000)=224:?((F+1)*(40+D*40)+
   &7000—2)=224
410 FOR T=1 TO 37+D*40:?(F*(40+D*40)
   +&7000+T)=255:NEXT:NEXT
420 ?A=4
430 J=RND(4)—1:G=J
440 B=A+S(J)*2:IF ?B=255 THEN ?B=
   J:?(A+S(J))=32:A=B:GOTO 430
450 J=(J+1)*—(J<3):IF J<>G THEN
   440
460 J=?A:?A=32:IF J<4 THEN A=A
   —S(J)*2:GOTO 430
470 ENDPROC
```

L'area di memoria scelta fa parte di quella dedicata allo schermo, cosicché si possono osservare le varie fasi della costruzione del labirinto (anche se solo in seguito la forma sarà riconoscibile). La linea

Due labirinti di diversa difficolta sull'Acorn

370 assegna l'indirizzo di partenza in memoria e la 380 calcola il numero di passi in ogni direzione. Le linee 390 e 400 riempiono il contorno del labirinto col carattere 224 e la 440 riempie l'intera area centrale con il 255.

Le altre linee creano il percorso attraverso il labirinto, la cui direzione è casuale, controllando ogni volta le locazioni di memoria per evitare l'omissione di parti del labirinto. Dovunque il percorso si sposti, il valore 32 (carattere spazio) sostituisce il carattere 255 nella locazione di memoria.

La PROCvisione ispeziona il valore (W) di tutte le locazioni e visualizza un blocco bianco se W vale 224 o 255 e uno spazio se W vale 32. In altre parole, visualizza il labirinto stesso:

```
480 DEFPROCvisione
490 VDU19,1,2,0,0,0
500 VDU10,10,10
510 FOR Y=0 TO 24
520 FOR X=0 TO 38+D*40
530 W=?(&7000+X+Y*(40+D*40))
540 VDU W
550 NEXT
560 VDU32
570 NEXT
580 PRINTSTRING$(40+40*D,"□")
590 FOR T=1 TO 50+D*50
600 X=RND(36+D*40)+1:Y=RND(23)+3
610 PRINTTAB(X,Y)"□"
620 NEXT
630 COLOUR1
640 VDU31,X,Y,226
650 PRINTTAB(6,0)"TEMPO","PUNTI","VITE"
660 ENDPROC
```

Il labirinto è riprodotto sullo schermo dalle linee 510-580. Le quattro linee successive creano delle "isole", usando alcuni spazi disposti a caso per complicare il tracciato delle pareti. La linea 640 visualizza il tesoro e la 650 le intestazioni per il tempo, il punteggio e il numero di vite.

Si noti come D, il livello di difficoltà, compaia in tutti i calcoli, cosicché il labirinto si adatta sia a uno schermo a 40 che a 80 colonne.

SVOLGIMENTO DEL GIOCO

La sezione seguente concerne lo svolgimento del gioco: DEFPROCgioco sposta il personaggio e misura il tempo impiegato a raggiungere il tesoro.

```
670 DEFPROCgioco
680 MOVE X,Y:GCOL0,0:VDU224,8:GCOL0,1:
    VDU225
690 LX=X:LY=Y
700 *FX21
710 K$=INKEY$(1)
720 VDU4:PRINT'TAB(0,1)TIME,SC,M:VDU5
730 IF TIME>MT THEN PROCchiudifile
740 IF M<0 THEN ENDPROC
750 IF K$="" THEN 710
760 IF K$="Z" AND POINT(X-32+D*16,
    Y)<>3-D*2 THEN X=X-32+D*16
770 IF K$="X" AND POINT(X+32-D*16,
    Y)<>3-D*2 THEN X=X+32-D*16
780 IF K$="L" AND POINT(X,Y-32)<>3
    -D*2 THEN Y=Y-32
790 IF K$="P" AND POINT(X,Y+32)<>3
    -D*2 THEN Y=Y+32
800 VDU4
810 VDU31,X/(32-16*D),32-Y/32
820 A%=&87:H=(USR
    (&FFF4) AND &FF00)/&100:H=H+96
830 IF H=226 THEN PRINTTAB(RND(36+
    D*40)+1,RND(22)+4)CHR$(226):SC=SC
    +MT-TIME: TIME=0
840 VDU5
850 MOVE LX,LY:GCOL0,0:VDU224
860 ENDPROC
```

La maggior parte delle routine qui presentate sono già state esposte in Giochi al Computer 3 e ormai molte linee dovrebbero essere familiari. La linea 680 ripulisce lo sfondo e visualizza un omino verde nella posizione di partenza. Le linee 710 e da 750 a 790 ispezionano quale tasto viene premuto, in modo da calcolare la nuova posizione nella quale la linea 810 sposta il cursore. La linea 820 ricorre a una routine del sistema operativo per scoprire il codice ivi contenuto e il risultato viene inserito in H. Se H è uguale a 226 (abbiamo trovato il tesoro), viene visualizato un nuovo tesoro in una posizione casuale, il punteggio viene aumentato ed il tempo azzerato. Abbiamo adesso 30 secondi per la nuova caccia al tesoro. Se il tempo scade prima di averlo raggiunto, entra in funzione la linea 730 sottraendo una vita.

```
870 DEFPROCchiudifile
880 M=M-1
890 VDU4:PRINTTAB(X/(32-16*D),32-Y/
    32)CHR$(227);
900 FOR DE=1 TO 2000:NEXT
910 VDU8,225
920 VDU5:TIME=0
930 ENDPROC
```

Questa procedura toglie una vita, quindi, per manifestare l'avvenuto "decesso", la linea 890 visualizza una croce e, dopo una breve pausa, la linea 910 fa comparire il personaggio successivo. Quando si perde la terza vita, PROCfine elimina il personaggio, l'autorepeat e il cursore grafico:

```
940 DEFPROCfine
950 MOVE X,Y:GCOL0,0:VDU255
960 *FX12,0
970 VDU 4
980 FOR DE=1 TO 2000:NEXT
990 ENDPROC
```

Infine, è visualizzato il punteggio finale e offerta al giocatore un'altra partita.

Per ottenere labirinti casuali sul Dragon e sul Tandy si potrebbe semplicemente usare lo schermo testuale, ma in tal caso essi risulterebbero troppo semplici, considerate le grosse dimensioni e lo scarso numero dei blocchi a disposizione.

Abbiamo quindi preferito utilizzare lo schermo per la grafica ad alta risoluzione e, benché il programma risulti più complesso di uno concepito per lo schermo testuale, offre il vantaggio di poter creare una gamma di labirinti di diversa complessità e con più livelli di difficoltà.

Si immagini il percorso casuale come costituito da una serie di blocchi quadrati: per un labirinto semplice vanno scelti blocchi di grossa dimensione, ma per uno di maggiore complessità se ne devono scegliere di più piccoli.

La prima sezione del programma inizializza le variabili e prepara in generale il computer per il disegno dei labirinti casuali. Lo si trascriva, ma senza eseguire ancora il RUN (si otterrebbero un errore UL, ossia "Linea non definita", quando il programma cerca invano di andare alla linea 1000).

```
10 PMODE4,1
20 CLS:PRINT@193,"LIVELLO DI DIFFICOLTA
   (0-5)";
30 L$=INKEY$:IFL$<"0"ORL$>"5"THEN
```

```
    30
40 BS=12-VAL(L$):NX=2*INT(.5+
   128/BS):NY=2*INT(.5+96/BS)
50 SX=250-BS*NX:SY=190-BS*NY
60 DIMP(NX,NY),A(5),B(5)
70 PCLS5:DRAW"S"+STR$(INT(8.5-4*VAL
   (L$)/5))+"CØBMØ,ØBR2BDNFNGD3NFG"
80 GET(Ø,Ø)-(BS-1,BS-1),A,G
90 GET(1Ø,1Ø)-(BS+9,BS+9),B,G:COLOR5,
   Ø
100 CLS:PRINT@228,"STO CREANDO UN
   LABIRINTO DI LIVELLO□";L$
110 GOSUB1000
120 GOTO 120
```

La linea 1Ø predispone il computer all'uso del PMODE4 nel corso del programma. A questo livello non è abilitato lo schermo ad alta risoluzione, perciò tutto quanto, per ora, appare sullo schermo testuale. La linea 2Ø visualizza il messaggio "LIVELLI DI DIFFICOLTÀ (Ø×5)". L$ è il livello scelto dal giocatore: il valore numerico L$ (si veda VAL (L$) alla linea 4Ø) regola le dimensioni del blocco e di conseguenza la larghezza del percorso e la complessità del labirinto. INKEY$ alla linea 3Ø consente al giocare l'immissione di una sola cifra, prima che il programma continui, senza dover premere ENTER. Alla linea 4Ø, BS è la dimensione in pixel del blocco, che può variare da 7 a 12 pixel.

NX è il numero di blocchi sullo schermo in larghezza, NY in altezza.

Prima di disegnare il labirinto sullo schermo, il computer ne definisce la fisionomia e inserisce le relative informazioni nella matrice P, dimensionata alla linea 6Ø. Le matrici A e B contengono rispettivamente la sagoma di un personaggio e uno spazio vuoto per la sua animazione.

Il personaggio viene disegnato dalla linea 7Ø. A pagina 185 si è visto come sia possibile variare la dimensione dei disegni: in questo programma la dimensione del personaggio viene proporzionata alle dimensioni del percorso nei labirinti.

Una volta disegnato, il personaggio viene trasferito nella matrice A (linea 8Ø), mentre la linea 9Ø riempie la matrice B di bianco. Fino ad ora, non si è mai presentata la necessità di riempire una matrice di caratteri spazio, dato che ciò equivarrebbe ad una matrice vuota. Stavolta, però, il colore del carattere spazio deve abbinarsi a quello del percorso.

Il comando COLOR (linea 9Ø) è necessario affinché, più avanti nel programma, il labirinto venga tracciato nel colore corretto o potremmo ritrovarci con un labirinto nero su sfondo nero!

La linea 1ØØ avverte il giocatore che il labirinto è in costruzione e, poco dopo, esso compare sullo schermo.

IL DISEGNO DEL LABIRINTO

Ecco la subroutine per la costruzione del labirinto, richiamata dalla linea 11Ø, dopo la quale lanciare il programma:

```
1000 FORJ=ØTO NX:P(J,NY)=6:P(J,Ø)=6:
   NEXT
1010 FORJ=ØTO NY-2:P(Ø,J)=6:P(NX,J)
   =6:NEXT
1020 X=2:Y=2:LX=2:LY=2
1030 J=RND(4)-1:G=J
1040 Y=LY+2*((J=Ø)-(J=2)):X=LX+
   2*((J=3)-(J=1))
1050 IFP(X,Y)=Ø THENP(X,Y)=J+1:P((X
   +LX)/2,(Y+LY)/2)=5:LX=X:LY=Y:
   GOTO1030
1060 J=(J+1)AND3:IF J<>G THEN1040
1070 J=P(LX,LY)-1:P(LX,LY)=5:IF J<
   4 THEN LX=LX-2*((J=3)-(J=1)):LY
   =LY-2*((J=Ø)-(J=2)):GOTO1030
1Ø8Ø FORJ=ØTOØ2Ø:P(2+2*RND((NX-3)/2),
      1+RND(NY-3))=5:P(1+RND
      (NX-3), 2+2*RND((NY-3)/2))=
      5:NEXT
   1Ø9Ø SCREEN 1,1:PCLS
   11ØØ FORJ=2TO NX-2:FORK=
      2TONY-2
   111Ø IFP(J,K)=5 THEN LINE (J*BS+
      SX,K* BS+SY)-((J+1)*BS+SX
      -1,(K+1)*BS+SY-1),PSET,BF
   112Ø NEXTK,J:RETURN
```

Le linee da 1ØØØ a 1Ø8Ø creano il labirinto nella memoria del computer, inserendolo nella matrice P: ogni elemento della matrice corrisponde a un blocco nel labirinto. La subroutine memorizza un 5 ogni volta che c'è un blocco di percorso e uno Ø per ogni blocco di parete. Quando P contiene l'intero labirinto, la linea 1Ø9Ø abilita lo schermo ad alta risoluzione per disegnarlo. Le linee 11ØØ-112Ø visualizzano il

labirinto sullo schermo esaminando il contenuto di P: in corrispondenza di ogni 5 è visualizzato un quadrato bianco.

MOVIMENTO, TESORO, VITE

Adesso ci occorre un programma di gioco che utilizzi i labirinti. Anche la seguente sezione di programma va trascritta, ma non ancora eseguita, poiché richiama una subroutine non ancora immessa. Sul Tandy, alla linea 17Ø, si usi 251 anziché 239, alla 18Ø 253 invece di 247, alle linee 19Ø e 2ØØ si usi 247 anziché 233:

```
120 X=2:Y=2:LX=2:LY=2:TI=800:LI=3
130 TIMER=0
140 X1=1+RND(NX-3):Y1=1+RND(NY
    -3):IF P(X1,Y1)=5THENP(X1,Y1)=7:
    DRAW"S4C0BM"+STR$(SX+X1*BS)+
    ","+STR$(SY+Y1*BS)+"BFR5D5L3U3
    RD" ELSE140
150 X1=X*BS+SX:Y1=Y*BS+SY
160 PUT(X1,Y1)-(X1+BS-1,Y1+BS-1),
    A,PSET
170 IFPEEK(338)=239 THEN Y=Y-1
180 IFPEEK(342)=247 THEN Y=Y+1
190 IFPEEK(340)=223 THEN X=X-1
200 IFPEEK(338)=223 THEN X=X+1
210 IFP(X,Y)=7 THENF=1:P(X,Y)=5:GOTO
    230
220 IFP(X,Y)<>5 THEN X=LX:Y=LY:
    GOTO170
230 IF X<>LX OR Y<>LY THENPUT(X1,
    Y1)-(X1+BS-1,Y1+BS-1),B,PSET:
    LX=X:LY=Y:FORP=1TO68:NEXT
240 IFF=1 THENF=0:SC=SC+(TI-
    TIMER):TI=TI-10:GOTO130
250 IFTIMER>TI THENGOSUB500:IFLI<1
    THEN100
260 GOTO150
```

La linea 12Ø, che rimpiazza quella precedentemente immessa, contiene le variabili necessarie al calcolo della posizione del personaggio. La coppia X, Y individua la sua posizione corrente, mentre LX, LY quella precedente. Ma, poiché la larghezza del percorso può variare, i valori vengono opportunamente adattati, prima della visualizzazione. TI è il tempo massimo, stabilito in 16 secondi, per raggiungere il tesoro. Se il tempo scade, si perde una vita. All'inizio del gioco, il giocatore possiede tre vite, cioè LI=3.

La linea 13Ø azzera il contatempo, prima che la 14Ø selezioni una posizione casuale per il tesoro. Per garantire che questo si trovi sul percorso, viene esaminato l'elemento corrispondente in P e, in caso affermativo, il valore nella matrice cambia da 5 a 7. L'ultima parte della linea disegna il tesoro nel labirinto.

La posizione del personaggio è calcolata alla linea 15Ø, in base alla larghezza del percorso, cioè BS, quella di un singolo blocco. La linea 16Ø pone il personaggio sullo schermo in tale posizione. Le linee dalla 17Ø alla 22Ø non dovrebbero richiedere alcuna spiegazione; sono le ormai note istruzioni per muovere il personaggio. Quest'ultimo non deve attraversare le pareti e di ciò si occupa la linea 22Ø.

La linea 21Ø verifica se è stato raggiunto il tesoro (il corrispondente valore in P varrebbe 7). Se questo è il caso, l'evento è segnalato da F, che viene posta a 1. La linea 23Ø genera il movimento del personaggio, cancellandolo dalla sua ultima posizione e visualizzandolo nella nuova.

La linea 24Ø calcola il punteggio, se si è trovato il tesoro. Il tempo massimo viene diminuito di 1Ø secondi, F viene azzerata e il programma torna indietro a riazzerare il timer. Il programma prosegue, visualizzando un tesoro in un nuovo punto, ma senza spostare il personaggio dalla posizione in cui si trova. Se il giocatore non trova il tesoro nel tempo massimo, la linea 25Ø richiama la subroutine che inizia alla linea 5ØØ.

Se il personaggio non ha trovato il tesoro, ma ha ancora tempo disponibile, la linea 26Ø costringe il programma a calcolare la sua nuova posizione.

VISUALIZZAZIONE DEI PUNTI

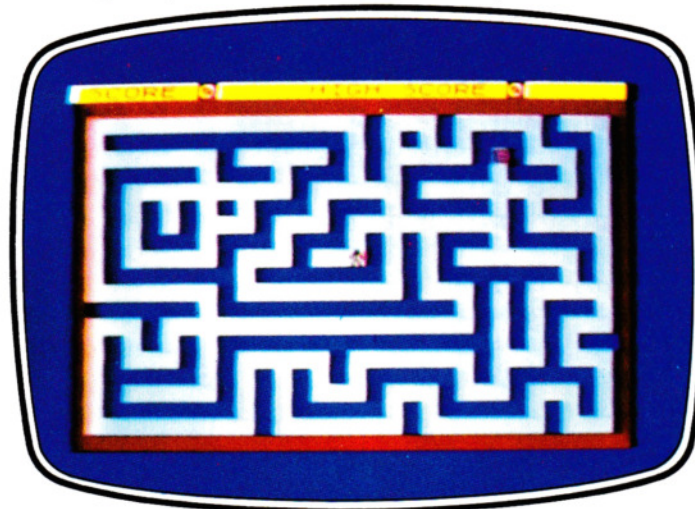
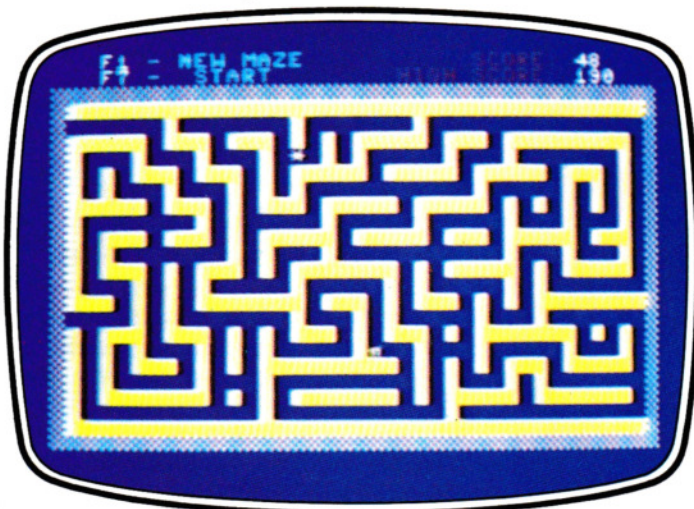
Quest'ultima subroutine visualizza i punti e il numero di vite rimanenti dopo un tentativo fallito:

```
500 CLS:SCREEN0,0:LI=LI-1
510 PRINT@106,"LIVELLO=";L$
520 IFLI>0 THEN PRINT@202,"VITE=";LI
530 PRINT@298,"PUNTI=";SC
540 IF LI>0 THENFORJ=1TO6000:NEXT:
    TIMER=0:SCREEN1,1:RETURN
550 PRINT@358,"UN ALTRO GIOCO (S/N)?"
560 A$=INKEY$:IFA$<>"S"ANDA$<>
    "N" THEN560
570 IFA$="S" THEN RUN
580 END
```

Adesso possiamo eseguire il programma. Quando si perde una vita, viene riattivato lo schermo testuale, con SCREEN Ø,Ø. La linea 5ØØ decrementa il numero di vite rimanenti. Le linee da 15Ø a 53Ø visualizzano il livello di difficoltà, il numero di vite rimaste, (se il gioco continua) e il punteggio. Finché si hanno vite, la linea 54Ø inserisce una pausa prima di riazzerare il timer, riabilitare lo schermo ad alta risoluzione e tornare al programma chiamante mediante la RETURN.

Le linee dalla 55Ø alla 58Ø offrono al giocatore un'altra partita, quindi o fermano il programma o lo rilanciano. Alla linea 57Ø si usa un RUN, azzerando così P.

Nel labirinto del Commodore si muove un pirata

Il pirata nel labirinto a colori dello Spectrum

SVISCERIAMO LE STRINGHE

- CONFRONTO E RIORDINAMENTO DI STRINGHE
- SEGMENTAZIONE DI STRINGHE
- QUANTO È LUNGA UNA STRINGA?
- USO DI STRINGHE NEL WORD PROCESSING

Le stringhe si usano in ogni tipo di programma, in pratica ovunque occorra elaborare qualcosa di più che semplici numeri. Ecco alcuni metodi per utilizzarle appieno.

Una stringa è costituita da una sequenza di caratteri. Questi possono essere lettere, numeri, punteggiatura o qualsiasi simbolo della tastiera, che si avvicendano in qualsiasi ordine.

Normalmente, una stringa contiene una qualche informazione, ma talvolta anche più elementi vengono raccolti in una sola stringa. Per esempio, "PAOLO ROSSI 241067 C" consiste di nome, cognome, data di nascita e stato civile: quattro elementi di un'informazione. Se poi consideriamo la data come ogni giorno, mese e anno, il numero degli elementi sale a sei.

A volte può essere utile suddividere (o segmentare) una stringa per estrarne uno degli elementi di informazione, come la data, nell'esempio sopra. Altre volte è necessario concatenare più stringhe assieme ed altre ancora misurarne la lunghezza o calcolare il valore di ogni sua parte numerica. Tutto ciò si può facilmente ottenere con poche parole chiave del BASIC.

Sommare stringhe, o concatenarle è la cosa più semplice, perché basta usare il simbolo '+': se A$ contiene "CIAO" e B$ "MAMMA", allora A$+B$ vale "CIAO MAMMA". Concatenare due stringhe vuol dire unirle, non esattamente sommarle: con "439"+"241" il risultato è "439241" (non 680!).

CONFRONTO DI STRINGHE

Il BASIC può confrontare, oltre che unire tra loro, stringhe per vedere se sono uguali, come in questo indovinello:

```
10 G=1: GOTO RND(6)*10+10
20 B$="MELA": GOTO 80
30 B$="ARANCIA": GOTO 80
40 B$="BANANA": GOTO 80
50 B$="LIMONE": GOTO 80
60 B$="FRAGOLA": GOTO 80
70 B$="ANANAS"
80 CLS: PRINT"SONO UN FRUTTO, QUALE
   FRUTTO□";
90 INPUT A$
100 IF A$=B$ THEN GOTO 170
110 G=G+1
120 PRINT"SBAGLIATO!"
130 FOR J=1 TO 2000
140 NEXT J
150 CLS
160 GOTO 90
170 IF G=1 THEN PRINT "HAI INDOVINATO
   ALLA PRIMA" ELSE PRINT"HAI
   INDOVINATO IN□";G;"□TENTATIVI"
```

Il programma funziona anche sullo ZX81, purché si suddividano le linee con istruzioni multiple in linee separate:

```
10 LET G=1: GOTO INT (RND*6)*10+10
20 LET B$="MELA": GOTO 80
30 LET B$="ARANCIA": GOTO 80
40 LET B$="BANANA": GOTO 80
50 LET B$="LIMONE": GOTO 80
60 LET B$="FRAGOLA": GOTO 80
70 LET B$="ANANAS"
80 CLS: PRINT "SONO UN FRUTTO, QUALE
   FRUTTO?"
90 INPUT A$
100 IF A$=B$ THEN GOTO 160
110 LET G=G+1
120 PRINT "SBAGLIATO!"
130 FOR J=1 TO 200
140 NEXT J
150 GOTO 90
160 IF G=1 THEN PRINT "HAI INDOVINATO
   ALLA PRIMA": STOP
170 PRINT "HAI INDOVINATO
   IN□";G;"□TENTATIVI"
180 STOP
```

```
10 G=1: ON RND(6) GOTO 20,30,40,50,60,
   70
20 B$="MELA": GOTO 80
30 B$="ARANCIA": GOTO 80
40 B$="BANANA": GOTO 80
50 B$="LIMONE": GOTO 80
60 B$="FRAGOLA": GOTO 80
70 B$="ANANAS"
```

```
80 CLS: PRINT"SONO UN FRUTTO, QUALE
   FRUTTO?"
90 INPUT A$
100 IF A$=B$ THEN GOTO 160
110 G=G+1
120 PRINT"SBAGLIATO!"
130 FOR J=1 TO 1000
140 NEXT J
150 GOTO 90
160 IF G=1 THEN PRINT"HAI INDOVINATO
   ALLA PRIMA" ELSE PRINT"HAI
   INDOVINATO IN";G;"TENTATIVI"
```

```
10 G=1:ON INT(RND(1)*6)+1 GOTO 20,30,
   40,50,60,70
20 B$="MELA":GOTO80
30 B$="ARANCIA":GOTO 80
40 B$="BANANA":GOTO 80
50 B$="LIMONE":GOTO 80
60 B$="FRAGOLA":GOTO 80
70 B$="ANANAS"
80 PRINT "♥SONO UN FRUTTO, QUALE
   FRUTTO?"
90 INPUT A$
100 IF A$=B$ THEN GOTO 155
110 G=G+1
120 PRINT "SBAGLIATO!"
130 FOR J=1 TO 2000
140 NEXT J
150 GOTO 90
155 IF G=1 THEN PRINT "HAI INDOVINATO
   ALLA PRIMA":END
160 PRINT "HAI INDOVINATO
   IN";G;"TENTATIVI"
170 END
```

La linea 10 assegna al contatore il valore 1, poi 'lancia un dado', per scegliere un frutto: le scelte sono contenute nelle linee da 20 a 70. A qualsiasi di queste linea vada, il computer deposita il nome del frutto nella stringa B$, passando poi alla linea 80. Adesso tocca a noi indovinare il frutto scelto immettendo un nome che finisce in A$. Questa viene quindi confrontata con B$ e, se coincidono alla perfezione, il computer visualizza "HAI INDOVINATO ALLA PRIMA" o "HAI INDOVINATO AL (ennesimo) TENTATIVO". Se A$ non è uguale a B$, il computer visualizza "SBAGLIATO!" e si passa a un nuovo tentativo, finché la risposta non è esatta. Non basta indovinare il frutto: se lo si scrive in modo scorretto, la condizione A$=B$ non è riconosciuta per vera. È necessario che *tutto*, nelle due stringhe coincida: lettere, spazi, punteggiatura e numero di battute.

Il confronto tra stringhe è utilissimo per controllare gli input da tastiera, con linee del tipo:

```
IF A$="SI" THEN PRINT "SEI SICURO?"
```

Si noti che la condizione non è rispettata se si scrive "si" in lettere minuscole.

RIORDINAMENTO SU STRINGHE

Le stringhe si possono confrontare anche usando i segni di diseguaglianza ⟨ e ⟩. Ecco un esempio:

```
IF A$<B$ THEN PRINT "IL PRIMO È□";A$
```

Qui la condizione A$ ⟨B$ è valida se la stringa contenuta in A$ viene prima di quella in B$ secondo l'ordine alfabetico. Ma attenzione! Il computer ordina alfabeticamente in base al codice ASCII di ogni lettera, presa una alla volta: il codice di A è 65, quello di Z è 90. Però, anche le lettere minuscole hanno il loro codice ASCII: quello di a è 97 e quello si z è 122. Tutte le stringhe che iniziano con maiuscole, per il computer, vengono prima di quelle con le minuscole. Ancora peggio: numeri, segni di interpunzione, spazi e altri segni hanno tutti un codice ASCII, per cui la classificazione di stringhe segue un ordine tutto particolare. Tuttavia, è possibile ordinare alfabeticamente più stringhe, mediante '⟨' e '⟩'. Una routine di questo tipo, detta *bubble sort* viene presentata a pagina 216.

SEGMENTAZIONE DI STRINGHE

Da una stringa può essere estratto un singolo carattere o un'intera sequenza. Su Dragon, Tandy, Vic, Commodore e Acorn lo si fa con le funzioni LEFT$, RIGHT$ e MID$. Lo Spectrum usa tecniche diverse, esposte a parte.

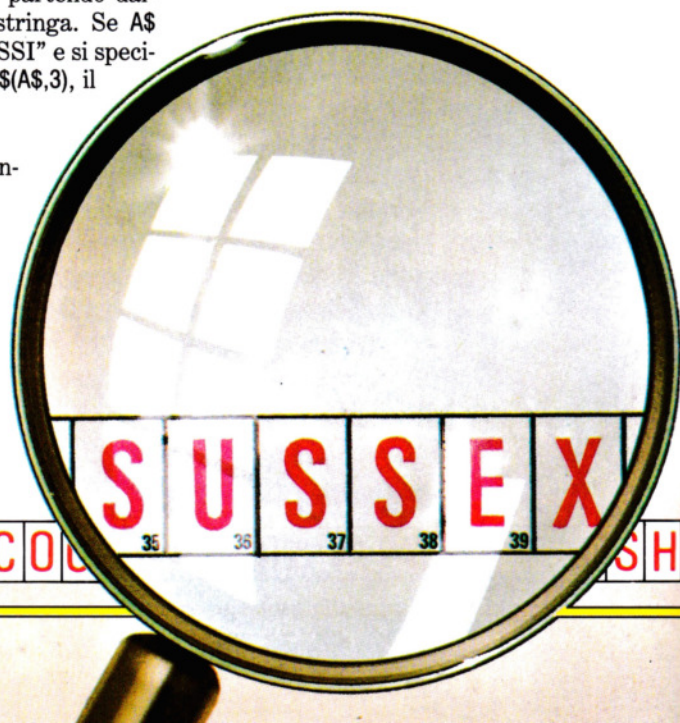
LEFT$ (A$, numero) estrae da A$ il numero di caratteri specificato, partendo dall'estremità sinistra della stringa. Se A$ contiene "DR. PAOLO ROSSI" e si specificano tre caratteri in LEFT$(A$,3), il risultato è "DR.".

RIGHT$ ha un funzionamento identico, ma parte invece dall'altra estremità della stringa, la destra: RIGHT$(A$,5) darà dunque "ROSSI". Con MID$ occorre specificare due numeri, la posizione da cui iniziare (partendo da sinistra) e il numero di caratteri da estrarre.

Per esempio, MID$(A$,5,7) parte dal quinto carattere ed estrae 7 caratteri, ossia "PAOLO R". Se si specifica soltanto un numero, tipo MID$(A$,3), si ottengono tutti i caratteri dal terzo in poi.

Sullo Spectrum, la segmentazione delle stringhe è ancora più semplice. Basta una sola funzione, secondo il formato: A$(numero TO numero). A$ identifica la stringa da segmentare e i due numeri sono l'inizio e la fine del segmento. Con la stessa stringa A$="DR PAOLO ROSSI", A$(1 TO 2) estrae "DR", A$(4 TO 8) dà "PAOLO" e A$(10 TO 14) dà "ROSSI". Non è necessario specificare ambedue i numeri: omettendo il primo lo Spectrum parte dall'inizio, mentre omettendo il secondo estrae tutti i caratteri restanti fino al termine della stringa.

Ecco un programma che con LEFT$, MID$ e LEN (vedi sotto) prepara un gioco di anagrammi per due giocatori. Il primo immette una parola, che il computer anagramma e poi visualizza, il secondo deve indovinarla.

```
10 CLS
20 PRINT@65, "ANAGRAMMI"
30 PRINT@161,"PAROLA DA
   RIMESCOLARE?"
40 A$=INKEY$:IF A$="" THEN 40
43 IF A$=CHR$(13) THEN 55
46 IF A$<"□" THEN 40
49 W$=W$+A$:GOTO 40
55 PAROLA$=W$
70 CLS
80 FOR N=LEN(W$) TO 1 STEP-1
90 M=RND(N)
100 A$=A$+MID$(W$,M,1)
110 W$=LEFT$(W$,M-1)+MID$(W$,M+
   1)
```

Mettiamo in ordine le stringhe

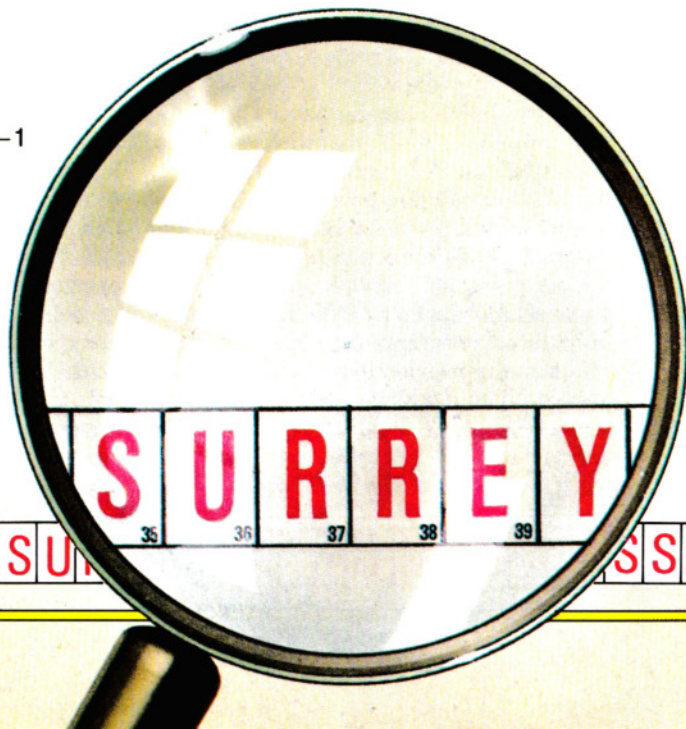
Se vediamo un computer estrapolare, come per magia, un indirizzo fra centinaia di voci separate e presentarlo sullo schermo, viene quasi da pensare che la macchina legga le varie voci e decida il da farsi. Non lasciamoci ingannare. Il computer è stato istruito per estrarre una determinata sequenza di caratteri: se la stringa contiene sciocchezze, sciocchezze otterremo. Per esser certi di estrarre la medesima informazione da ogni stringa, occorre assicurarsi che essa sia stata collocata al posto giusto durante l'immissione. In campo professionale si ricorre spesso all'impiego di schede, onde assegnare un formato standard a tutte le informazioni immesse. Le schede sono utilissime e facili da realizzare: basta allineare una serie di riquadri per i caratteri, corredati da opportune intestazioni e da indicazioni sul punto iniziale e terminale di ogni elemento dell'informazione. Un semplice esempio è a pagina 201.

```
120 NEXT N
130 PRINT@65,"L'ANAGRAMMA È□"A$
140 PRINT@129,"QUAL È LA PAROLA
    ORIGINALE?"
160 INPUT PROVA$
170 G=G+1
180 IF PROVA$<>PAROLA$ THEN
    PRINT"□ERRATO, RIPROVA":GOTO 160
190 PRINT:PRINT"□BRAVO!"
200 IF G=1 THEN PRINT"□INDOVINATO
    ALLA PRIMA" ELSE
    PRINT"□INDOVINATO IN";G;"TENTATIVI"
210 PRINT@480;"□ANCORA UNA PAROLA
    (S/N)?"
220 A$=INKEY$: IF A$<>"S" AND A$<
    >"N" THEN 220
230 IF A$="S" THEN RUN
240 END
```

```
10 CLS
20 PRINT '"ANAGRAMMI"
30 PRINT ''"QUAL È LA PAROLA DA
   RIMESCOLARE"
40 VDU 21
50 INPUT W$
55 WORD$=W$
60 VDU6
70 CLS
80 FOR N=LEN(W$) TO 1 STEP -1
90 M=RND(N)
100 A$=A$+MID$(W$,M,1)
110 W$=LEFT$(W$,M-1)+MID$(W$,M+
    1)
120 NEXT N
130 PRINT '"L'ANAGRAMMA È□";A$
140 PRINT '"QUAL È LA PAROLA
    ORIGINALE?"
160 INPUT INDO$
170 G=G+1
180 IF INDO$<>PAROLA$PRINT
    "SBAGLIATO, RIPROVARE":GOTO 160
190 PRINT '"ESATTO!"
200 IF G=1 PRINT "AL PRIMO TENTATIVO"
    ELSE PRINT "INDOVINATA,
    IN□";G;"□TENTATIVI"
210 PRINT ''"ANCORA UNA PAROLA (S/N)?"
220 A$=GET$:IF A$<>"S" AND A$<
    >"N" THEN 220
230 IF A$="S" THEN RUN
240 END
```

Per lo ZX81 si scrivano tutte le variabili in maiuscolo, si omettano le linee 40 e 60 e gli apostrofi dopo le istruzioni PRINT. Si separino le linee con istruzioni multiple:

```
10 CLS: LET a$="": LET g=0
20 PRINT '"ANAGRAMMI"
30 PRINT ''"QUAL È LA PAROLA DA
   RIMESCOLARE"
40 POKE 23609,20: POKE 23658,8: POKE
   23624,63
50 INPUT w$
55 LET s$=w$
60 POKE 23624,56
70 CLS
80 FOR n=LEN w$ TO 1 STEP -1
90 LET m=INT (RND*n)+1
100 LET a$=a$+w$(m)
110 LET w$=w$(TO m-1)+
    w$(m+1 TO)
120 NEXT n
130 PRINT '"L'ANAGRAMMA
    È□";a$
140 PRINT '"QUAL È LA
    PAROLA ORIGINALE?"
160 INPUT LINE g$
170 LET g=g+1
180 IF g$<>s$ THEN PRINT
    "SBAGLIATO, RIPROVARE": GOTO 160
190 PRINT '"ESATTO!"
195 IF g=1 THEN PRINT "AL PRIMO
    TENTATIVO!": GOTO 210
200 PRINT "IN□";g;"□TENTATIVI"
210 PRINT '"ANCORA UNA PAROLA (S/N)?"
220 LET a$=INKEY$: IF a$<>"S" AND a$
    <>"N" THEN GOTO 220
230 IF a$="S" THEN RUN
```

```
10 PRINT "♥"
20 PRINT "ANAGRAMMI"
30 PRINT "QUAL È LA PAROLA DA
   RIMESCOLARE"
50 INPUT ">■";W$
55 WO$=W$
70 PRINT "♥ ◆"
80 FOR N=LEN(W$) TO 1 STEP -1
90 M=INT(RND(1)*N)+1
100 A$=A$+MID$(W$,M,1)
110 W$=LEFT$(W$,M-1)+RIGHT$(W$,
    LEN(W$)-M)
120 NEXT N
130 PRINT "L'ANAGRAMMA È□";A$
140 PRINT "QUAL È LA PAROLA
    ORIGINALE?"
160 INPUT INDO$
170 G=G+1
180 IF INDO$<>WO$ THEN PRINT
    "SBAGLIATO, RIPROVARE":GOTO 160
190 PRINT "ESATTO!"
195 IF G=1 THEN PRINT "AL PRIMO
    TENTATIVO!": GOTO 210
200 PRINT "IN";G;"TENTATIVI"
210 PRINT "ANCORA UNA PAROLA (S/N)?"
220 GET A$:IF A$<>"S" AND A$<>"N"
    THEN 220
230 IF A$="S" THEN RUN
```

• Nella prima versione del sistema operativo 1:Ø per i micro BBC, la funzione INSTR non operava in modo del tutto corretto. Il problema sorgeva se la stringa di ricerca era più lunga della stringa sotto esame, causando talvolta addirittura la perdita del programma. Un simile disastroso evento può facilmente capitare per sbaglio. Ad esempio, in un programma che richieda l'input, in una stringa A$, di una direzione sulla bussola, si potrebbe usare INSTR("NESO"A$) per controllare l'esattezza della risposta. Ma guai ad immettere per intero "NORD"!

Il problema è stato ampiamente risolto sul micro BBC, ma, a scanso di equivoci, conviene controllare la lunghezza della stringa di ricerca con la funzione LEN, prima di usare INSTR.

Si cambino le linee 5Ø e 7Ø del programma Commodore in:

```
5Ø INPUT ">■";W$
7Ø PRINT "■■"
```

Quando si lancia questo gioco si noti come la parola digitata inizialmente non compare sullo schermo, in modo che il giocatore che deve riconoscerla non la veda. Per ottenere ciò ogni computer ha un metodo diverso. L'Acorn usa VDU 21 per disattivare l'uscita su schermo e poi VDU 6 per riattivarla, dopo che la parola è digitata. Il Commodore, il Vic e lo Spectrum, invece, scrivono la parola nello stesso colore dello sfondo, rendendola illegibile. Il Dragon e il Tandy usano INKEY$ che accetta un solo carattere della parola alla volta, senza visualizzarlo.

La routine di anagramma va dalla linea 8Ø alla 12Ø: i caratteri vengono estratti a caso dalla parola e aggiunti uno per volta ad A$, che poco a poco diventa l'anagramma. La linea 9Ø sceglie un numero M a caso tra 1 e la lunghezza della parola, poi la 1ØØ estrae la lettera di posizione M, accodandola ad A$. La 11Ø elimina dalla parola originale il carattere appena estratto, prendendo la parte a sinistra (fino a M) ed aggiungendola a quello che resta della parola dopo M. La parola si accorcia così di un carattere, ma al successivo giro del ciclo anche la variabile N è calata di uno, cosicché il numero M è ancora uguale alla lunghezza della parola.

Al termine, tutti i caratteri sono andati a formare l'anagramma che poi compare sullo schermo, per l'indovinello. Quando si indovina, compare l'informazione sul numero di partite giocate e l'invito a ricominciare. Non è difficile modificare il programma affinché le parole vengano lette da una serie di frasi DATA, anziché essere immesse ogni volta.

Si potrebbe poi aggiungere un contapunti, che assegni 1Ø al primo tentativo, 9 al secondo e via dicendo.

Un'altra applicazione della segmentazione di stringhe è nella manipolazione delle date. Queste vengono immesse in cifre, come 27/Ø3/51, cioè 27 marzo 1951, ma formano una sola stringa. È impossibile applicare le comuni leggi matematiche su una data così espressa, tuttavia può servire eseguire dei calcoli su singole parti di essa. Ad esempio, per calcolare l'età di una persona in un particolare giorno conoscendo la sua data di nascita, oppure i giorni trascorsi tra due date. Queste operazioni implicano l'anno, il mese e il giorno, prese come parti e a sé stanti.

LEFT$, RIGHT$ e MID$ (o l'equivalente metodo dello Spectrum) separano facilmente anno, mese e giorno di una data, poi con la funzione VAL (spiegata oltre) si convertono i segmenti di stringa ottenuti in un numero che si può sommare, sottrarre, moltiplicare o dividere.

LUNGHEZZE DI STRINGHE

Talvolta è utile conoscere la lunghezza di una stringa. Se, per esempio, è limitata la quantità di memoria riservata a un'informazione o la zona utilizzabile per una scritta sullo schermo, conoscere la lunghezza di un dato può tornare comodo.

LEN(A$) fornisce il numero di caratteri nella stringa A$ e non è una stringa, ma una funzione numerica, sulla quale si opera secondo le leggi dell'algebra. Per esempio, se A$ = "DR. PAOLO ROSSI", LEN(A$) = 15. Supponiamo di dover immettere un nome in un file, ma che il formato preveda soltanto un massimo di 11 caratteri. L'operatore potrebbe essere avvertito che occorre "tagliare" il dato nel seguente modo:

```
1Ø PRINT "NOME DELLA MATERIA"
2Ø INPUT A$
3Ø IF LEN(A$)>11 THEN PRINT
   "SPIACENTE, MASSIMO 11 CARATTERI":
   GOTO 1Ø
```

L'operatore potrebbe allora ridurre l'immissione a soltanto "PAOLO ROSSI".

In altri casi è più semplice troncare l'immissione automaticamente, per esempio con una linea come questa:

```
IF LEN(A$)>15 THEN LET A$=LEFT$(A$,
   15)
```

o sullo Spectrum

```
IF LEN(A$)>15 THEN LET A$=A$(TO 15)
```

DA STRINGHE A NUMERI

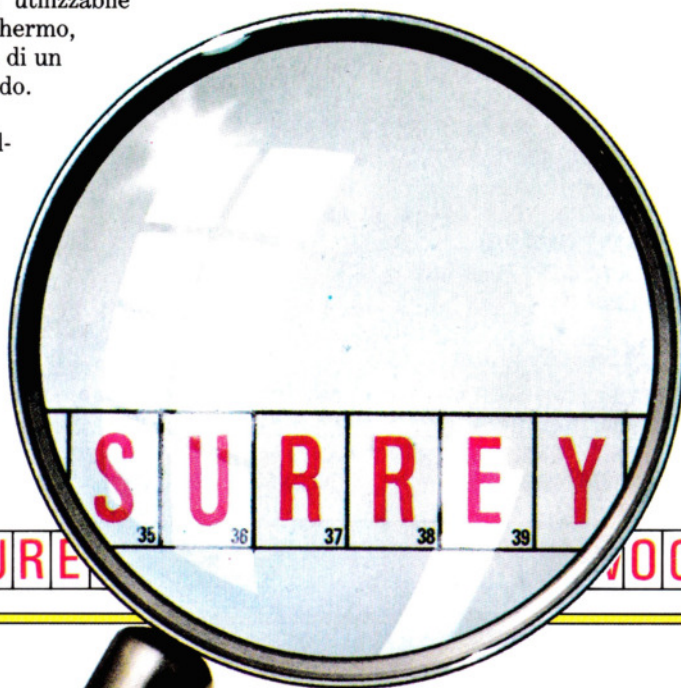
A volte le variabili stringa servono a correggere errori fatti alla tastiera. Per esempio, se si scrive un programma che include le linee:

```
1ØØ PRINT "IMMETTERE UN NUMERO"
11Ø INPUT A
```

il computer attende che si scriva un numero. Se per sbaglio si digita un carattere non numerico, la maggior parte dei computer (non l'Acorn), scombina la visualizzazione delle scritte sullo schermo, per far comparire un avviso di errore. Ma se si usa invece:

```
11Ø INPUT A$
```

l'operatore può digitare qualsiasi cosa e il computer l'accetta. Successivamente, si

H AMTYPIST AGENCYZ SURE OOD

può convertire la stringa in un numero e, a tale scopo, serve VAL(A$).

Questo sistema purtroppo non funziona sullo Spectrum e qualunque tentativo di calcolare il valore di una stringa di lettera causa sempre un errore. L'uso di VAL sullo Spectrum è limitato a stringhe composte interamente di numeri. Perciò VAL("1984") dà 1984 che è corretto, ma VAL("26/10/84") dà 0,03095, perché lo Spectrum usa VAL per valutare l'espressione 26:10:84.

I possessori di Spectrum possono perciò saltare al successivo paragrafo.

La funzione VAL su Dragon, Tandy, Acorn, Vic e Commodore valuta la parte numerica della stringa. Così se A$ rappresenta un numero, allora VAL(A$) ci fornisce tale numero, utilizzabile nel resto del programma. Se A$ non è un numero, allora VAL(A$) fornisce il valore 0. Una piccola subroutine potrebbe spiegare all'operatore dov'è l'errore, consentendogli un nuovo tentativo, senza uscire dal programma e perdere così i dati che ha sullo schermo.

Una cosa importante da ricordare su VAL è che essa valuta soltanto i numeri all'inizio della stringa: VAL("25 luglio") è uguale a 25, ma VAL ("luglio 25") è uguale a 0. Perciò attenzione!

La funzione VAL è utile per preordinare i numeri di una stringa in modi diversi. Se un insegnante conservasse i voti degli allievi in stringhe, A$="8 PAOLI" B$="PIERI" C$="7 BIANCHI" ecc., per avere la media della classe bastrebbe isolare i voti individuali con VAL. VAL(A$) dà 8, VAL(B$) dà 5 e VAL(C$) dà 7.

Allo stesso modo, si può usare VAL per ignorare le unità di misura immesse di seguito a un valore. Questo programma illustra il concetto:

```
100 LET A$="32 KG"
110 LET B$="110 KG"
120 PRINT "A$+B$="; A$+B$
130 PRINT "VAL(A$)+VAL(B$)=";VAL(A$)
    +VAL(B$)
140 END
```

DA NUMERI A STRINGHE

La funzione STR$ funziona all'opposto di VAL: cambia un numero in una stringa. Ne consegue la possibilità di adoperare tutte le funzioni di concatenamento e di segmentazione che non si possono usare direttamente sui numeri. La funzione STR$, come vedremo tra breve, ha numerose applicazioni.

Il seguente programma converte un numero decimale in binario. Sebbene i computer eseguano tutte le loro operazioni in numeri binari, il BASIC manipola soltanto i numeri decimali o, in alcuni casi, esadecimali. Perciò, quando in un programma in BASIC compare un numero binario, occorre trattarlo come una stringa:

```
10 PRINT "DA DECIMALE A BINARIO"
20 PRINT "NUMERO DECIMALE (INTERO)"
30 INPUT D
40 LET B$=""
50 LET B$=STR$(D-INT(D/2)*2)+B$
60 LET D=INT(D/2)
70 IF D<>0 THEN GOTO 50
80 PRINT "EQUIVALENTE IN BINARIO□";B$
```

Quando si immette un numero decimale positivo, la linea 40 pone B$ uguale a una stringa vuota o "nulla", che viene poi riempita di cifre man mano che il ciclo alle linee 50 e 80 esegue il computo.

La linea 50 è quella che effettivamente costruisce il numero binario: sottrae due volte il valore intero di metà del numero decimale dallo stesso numero decimale, per verificare se questo è dispari. In caso affermativo il risultato è 1, se invece è pari il risultato è 0 e queste sono le cifre binarie.

Sui computer dell'Acorn questa operazione si attua in modo più elegante, grazie alla funzione MOD:

```
50 B$=STR$(D MOD 2)+B$
```

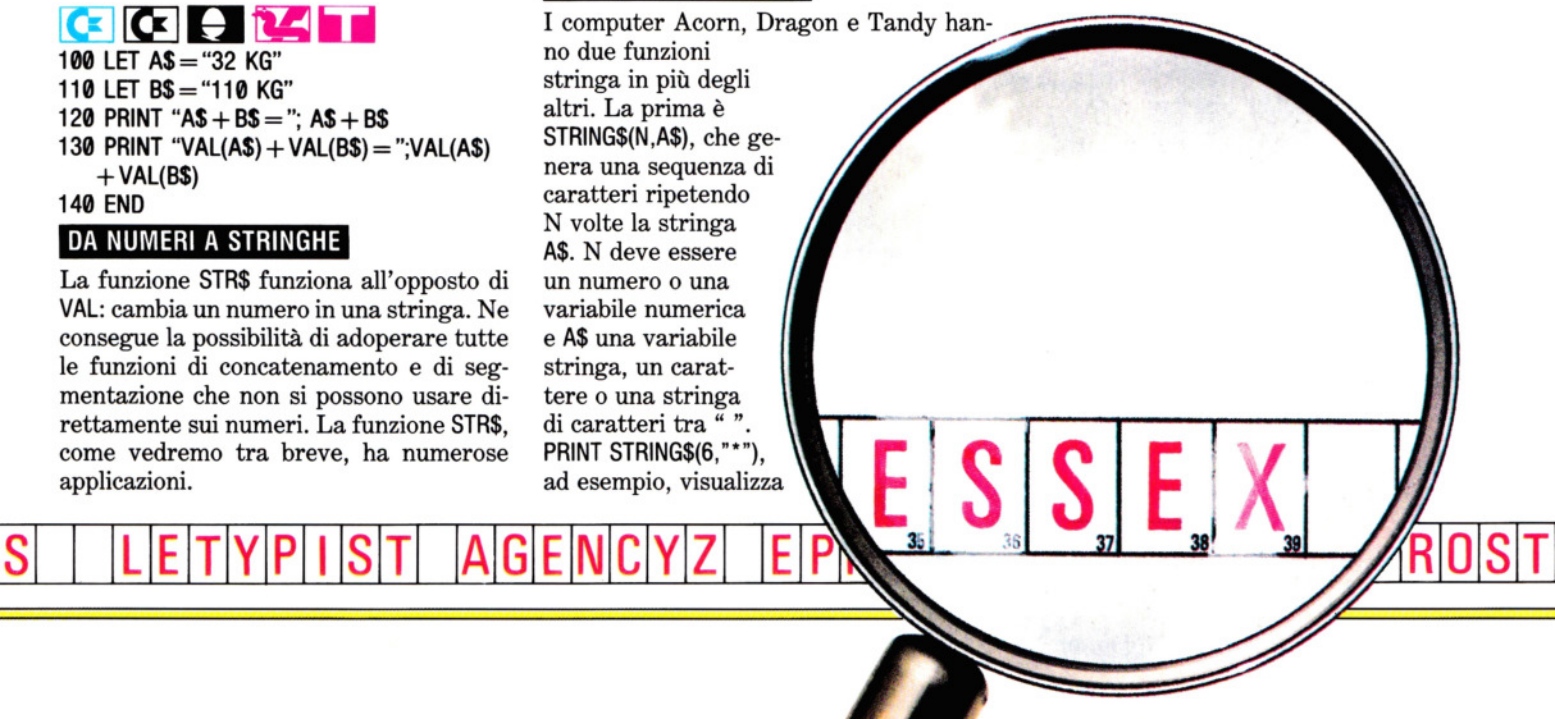
Le cifre binarie vengono convertite in una stringa mediante STR$ e il numero binario nasce dalla concatenazione di ogni nuova cifra con il resto della stringa.

LA FUNZIONE STRINGS

I computer Acorn, Dragon e Tandy hanno due funzioni stringa in più degli altri. La prima è STRING$(N,A$), che genera una sequenza di caratteri ripetendo N volte la stringa A$. N deve essere un numero o una variabile numerica e A$ una variabile stringa, un carattere o una stringa di caratteri tra " ". PRINT STRING$(6,"*"), ad esempio, visualizza ******. Sui computer Acorn, se definiamo A$ come "X-X" e N uguale a 3, la STRING$(3,A$) ci dà X-XX-XX-X. Questa funzione è comoda per ottenere linee o cornici decorative, partendo da un semplice modulo ripetitivo, ma anche in tutti quei casi nei quali una lunga stringa può essere ottenuta da una sequenza di stringhe più brevi.

Sul Dragon e sul Tandy, al contrario, STRING$(3,A$) ripete soltanto il *primo* carattere della stringa, ignorando il rimanente di A$.

Ecco due programmi (uno per il Dragon e il Tandy e uno per gli Acorn) che usano STRING$ per visualizzare una cornice decorativa intorno allo schermo. Può essere usato per decorare la prima pagina di un gioco:

```
10 CLS0
20 A$=CHR$(158)+STRING$(30,CHR$
   (156))+CHR$(157)
30 B$=CHR$(154)+CHR$(174)+STRING$
   (28,CHR$(172))+CHR$(173)+CHR$(149)
40 C$=CHR$(154)+CHR$(171)+STRING$
   (28,CHR$(163))+CHR$(167)+CHR$(149)
50 D$=CHR$(155)+STRING$(30,CHR$
   (147))+CHR$(151)
60 F$=CHR$(154)+CHR$(170)+STRING$
   (28,"□")+CHR$(165)+CHR$(149)
70 PRINTA$;
80 PRINTB$;
90 FOR K=1 TO 11
100 PRINTF$;
110 NEXT K
120 PRINTC$;
130 PRINTD$;
140 PRINT@233,"EHI, CIAO!";
150 GOTO 150
```

```
10 MODE1
20 VDU23;8202;0;0;0;
30 VDU19,0,4,0,0,0,19,2,2,0,0,0
40 COLOUR2
50 PRINTTAB(4,4)STRING$(8,"xoox")
60 FOR T=1 TO 11
70 PRINTTAB(4)"o"TAB(35)"o"
80 PRINTTAB(4)"x"TAB(35)"x"
90 NEXT
100 PRINTTAB(4)STRING$(8,"xoox")
110 COLOUR1
120 FOR T=1 TO 11 STEP 2
130 PRINTTAB(9,9+T)"o"TAB(21)"o"
140 PRINTTAB(9,10+T)"x"TAB(21)"x"
150 NEXT
160 PRINTTAB(10,10)STRING$(5,"xoox")
170 PRINTTAB(10,21)STRING$(5,"xoox")
180 COLOUR3
190 PRINTTAB(15,15)"EHI, CIAO!"
200 GOTO 200
```

Le versioni del Dragon e del Tandy utilizzano diversi caratteri grafici per disegnare la cornice. I blocchi hanno due diversi schemi di colore, giallo/nero e blu/nero. Per ottenere una cornice simmetrica i colori sono invertiti dall'alto al basso e da un lato all'altro. Vengono usate cinque stringhe: A$ e B$ per il lato superiore, F$ per i fianchi e C$ e D$ per il lato inferiore. La linea 150 evita che il messaggio OK ci rovini il disegno.

Il programma Acorn usa i normali caratteri della tastiera "XOOX" per fare la cornice. STRING$ serve per il lato superiore e quello inferiore; i fianchi invece sono disegnati più semplicemente con TAB. La linea 200 evita la comparsa del *prompt* del BASIC, che rovinerebbe il disegno.

LA RICERCA IN UNA STRINGA

INSTR è una funzione di ricerca, utile per individuare una stringa breve in una più lunga. Ad esempio, la si può usare per cercare una parola in una frase o una lettera in una parola.

Scrivendo INSTR(A$,B$), il computer ricerca B$ in A$ e fornisce la posizione dove ha individuato B$ per la prima volta nella stringa A$: PRINT INSTR("CIAO","A") visualizza 3. Se il computer non trova la stringa richiesta, il risultato è 0, come nel seguente esempio:

```
10 A$="CIAO!"
20 B$="W"
30 PRINT INSTR(A$,B$)
```

Il funzionamento di INSTR sugli Acorn è leggermente diverso da quello sul Dragon e sul Tandy. In emtrambi i casi viene specificato un numero tra le parentesi, ma sul Dragon e il Tandy esso compare per *primo*, INSTR(P,A$,B$), mentre sull'Acorn per *ultimo* INSTR(A$,B$,P). Il numero P ha comunque lo stesso significato: indica la posizione in A$ da cui si vuole iniziare la ricerca. A volte è utile iniziare la ricerca ad un certo punto di una stringa, anziché sempre all'inizio.

Supponiamo di voler trovare la posizione di tutte le "N" in una parola, ad esempio "ANTINCENDIO". Applicando INSTR una prima volta, si ottiene la posizione 2. Per trovare la successiva, occorre partire da P=2+1, cioè 3, in INSTR(P,"ANTINCENDIO","T") o INSTR ("ANTINCENDIO","T",P). Trovata la seconda N alla posizione 5, si pone P=6 per l'ultima. Se il valore di P non è specificato, la ricerca parte dal primo carattere della stringa.

Se in una stringa si cerca una stringa nulla, ossia "", il risultato è sempre 1.

Un altro uso di INSTR è nel controllo di input da tastiera. Supponiamo di poter scegliere un'opzione da un menu, usando la sola lettera iniziale. Le opzioni potrebbero essere;

```
10 PRINT "(S)tampa del testo"
20 PRINT "(R)egistrazione del testo"
30 PRINT "(L)ettura di un nuovo testo"
40 PRINT "(E)laborazione del testo"
50 PRINT "DIGITARE LA SCELTA"
60 INPUT A$
```

Un modo pratico per controllare se la lettera immessa è valida è quello di aggiungere la linea:

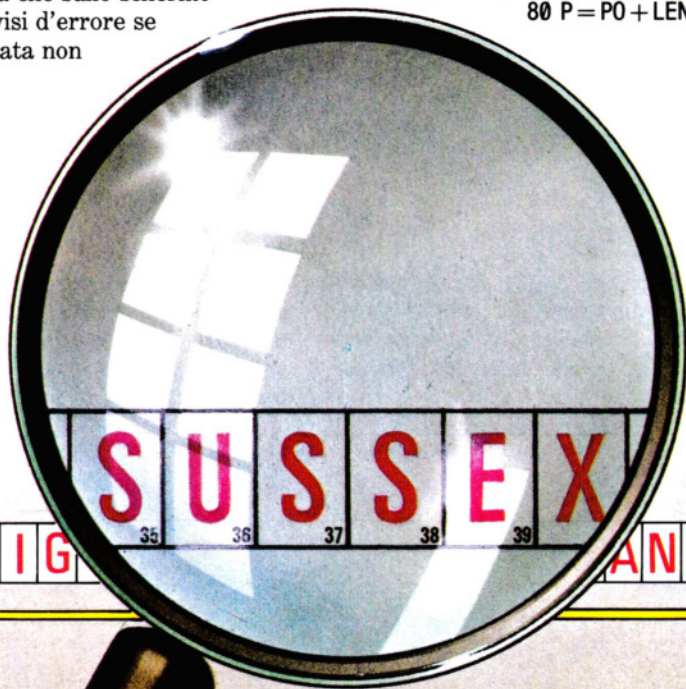
```
70 IF INSTR("SRLE",A$)=0 THEN GOTO 50
```

Con ciò si evita che sullo schermo compaiano avvisi d'errore se la lettera digitata non è valida.

WORD PROCESSING

Nell'elaborazione dei testi, ormai nota col termine *word processing*, viene fatto largo uso delle funzioni stringa. Può esser necessario, ad esempio, sostituire nell'intero testo una parola con un'altra (per correggere un errore di battitura): INSTR trova per noi tutte le occorrenze della parola, consentendoci la sostituzione con quella corretta. Se il computer non dispone di INSTR, esistono altri metodi, ma molto più lenti. Naturalmente, se la nuova parola ha una lunghezza diversa, tutto il testo va fatto scorrere, per lasciare spazio sufficiente all'inserimento. Il seguente programma mostra come fare:

```
10 INPUTLINE "IMMETTERE IL TESTO",T$
20 INPUTLINE "PAROLA DA SOSTITUIRE",W$
30 INPUTLINE "NUOVA PAROLA",NW$
40 P=1
50 pos=INSTR(T$,W$,P)
60 IF pos=0 THEN GOTO 100
70 T$=LEFT$(T$,pos-1)+NW$+RIGHT$
   (T$,LEN(T$)-pos-LEN(W$)+1)
80 P=pos+LEN(NW$)
90 GOTO 50
100 PRINT T$
110 GOTO 20
```

```
10 LINEINPUT "IMMETTERE IL TESTO?";T$
20 LINEINPUT "PAROLA DA
   SOSTITUIRE?";W$
30 LINEINPUT "NUOVA PAROLA?";NW$
40 P=1
50 PO=INSTR(P,T$,W$)
60 IF PO=0 THEN GOTO 100
70 T$=LEFT$(T$,PO-1)+NW$+RIGHT$
   (T$,LEN(T$)-PO-LEN(W$)+1)
80 P=PO+LEN(NW$)
```

```
90 GOTO 50
100 PRINT T$
110 GOTO 20
```

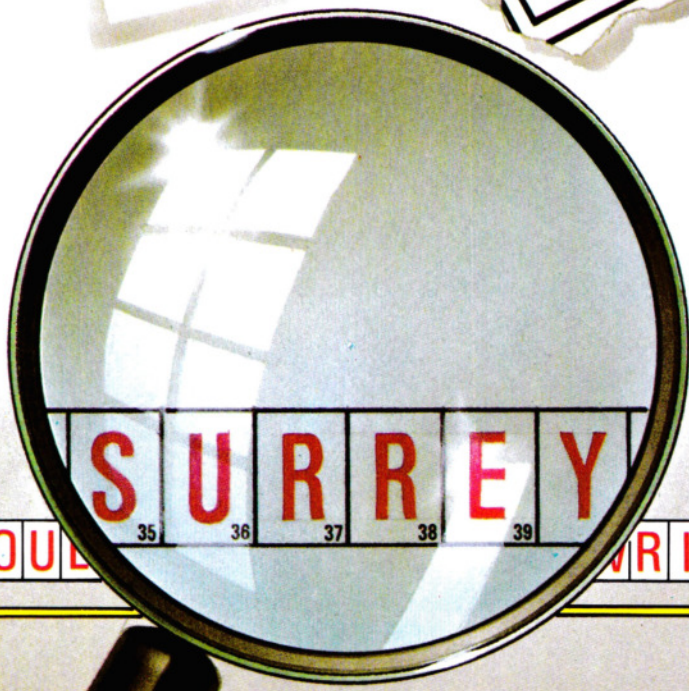
```
10 INPUT "IMMETTERE IL TESTO";T$
20 INPUT "PAROLA DA SOSTITUIRE";W$
30 INPUT "NUOVA PAROLA";NW$
35 P=0
40 P=P+1
50 A$=MID$(T$,P,LEN(W$))
60 IF A$<>W$ THEN 90
70 T$=LEFT$(T$,P-1)+NW$+RIGHT$
   (T$,LEN(T$)-P-LEN(W$)+1)
80 P=P+LEN(NW$)-1
90 IF P<LEN(T$) THEN GOTO 40
100 PRINT T$
110 GOTO 20
```

```
10 INPUT "IMMETTERE IL TESTO□"; LINE
   t$: PRINT t$
20 INPUT "PAROLA DA SOSTITUIRE□"; LINE
   w$: LET w=LEN w$
30 INPUT "NUOVA PAROLA□"; LINE n$: LET
   n=LEN n$
35 LET p=0
40 LET p=p+1
50 IF p+w-1>LEN t$ THEN GOTO 100
60 IF t$(p TO p+w-1)<>w$ THEN GOTO
   40
70 LET t$=t$(TO p-1)+n$+t$(p+w TO):
   GOTO 40
100 PRINT t$
110 GOTO 20
```

Si cominci con l'immissione di una frase semplice, poi si provi a sostituire qualche lettera o parola. È meglio se parole brevi (come 'io' o 'per') sono precedute e seguite da almeno uno spazio, altrimenti verrebbero sostituite anche ogni volta che capitano all'interno di altre parole (come in 'pioggia' o in 'perla'). Sul Commodore si immettano le parole di ricerca racchiudendole fra apici, ad esempio: "per".

La linea 40 pone P all'inizio del testo per l'operazione di ricerca. Le linee 50 e 60 trovano la prima occorrenza della parola o della lettera da sostituire e la 70 inserisce la nuova parola. Non fa che depositare in T$ il testo originario, fino a trovare la vecchia parola, aggiunge la nuova e poi il testo restante.

Il procedimento si ripete per tutte le occorrenze della parola, quindi la linea 100 visualizza il risultato.

Questo è un esempio molto semplice (i veri word processor sono molto più complicati), tuttavia illustra alcune applicazioni pratiche delle funzioni stringa.

COME SON FATTE LE MEMORIE

Sul corretto funzionamento della memoria di computer poggia quello dell'intero sistema. Ma cosa sono, come sono concepite e realizzate le memorie?

Entrare nel mondo della programmazione in codice macchina significa toccare il livello più profondo del proprio home computer. Non basta soltanto una conoscenza superficiale dei numeri esadecimali che si vanno scrivendo o dei numeri binari nei quali il computer li converte (vedi a pagina 156), occorre anche capire come lavora lo stesso computer. Non è necessario studiare come si collegano e si installano i chip, ma è bene capire qualcosa dell'architettura complessiva dell'apparecchio, le relazioni tra le varie parti del sistema e le funzioni di ciascuna di esse.

DI COSA È FATTA LA MEMORIA

La memoria degli home computer è contenuta in una serie di chip di silicio, ciascuno contenente migliaia di piccoli interruttori, che possono essere o accesi o spenti. Ogni interruttore equivale a una sola cifra in binario, ossia un *bit*. *Il circuito acceso rappresenta 1, spento Ø.*

All'interno del chip, questi piccoli circuiti sono organizzati in gruppi di otto e ogni gruppo è un *byte*, ossia un numero binario di otto bit o due cifre in hex. Non avrebbe però senso depositare numeri in questi circuiti senza sapere come andarli a ritrovare. Perciò a ogni locazione di memoria di otto bit viene assegnato un indirizzo. Per esempio, se la memoria totale disponibile è di 64K, servono 64 * 1Ø24 indirizzi (uno per locazione).

QUANTA MEMORIA?

In informatica, un K è approssimativamente analogo al 'K' che sta per Kilo, mille nel sistema metrico, ma 1ØØØ non va bene per essere convertito in binario o hex. Il numero hex più adatto è 4ØØ, che vale 1Ø24 in decimale e che è rappresentato in binario da 1 seguito da dieci zeri, cioè 1K.

Dovendo identificare locazioni di memoria, si potrebbero numerare da 1 a 65.536 in decimale, ma è più conveniente numerarle in hex da ØØØØ a FFFF, secondo il sistema usato dal computer. In tal modo si usa ogni possibile combinazione hex di quattro cifre, o due byte. Il numero hex di quattro cifre, assegnato a una locazione di memoria, è il suo indirizzo. Ogni locazione può essere *indirizzata*, ossia individuata per scrivervi o leggervi un valore, servendosi unicamente di un numero hex di quattro cifre.

Dei computer esaminati nel corso di INPUT, solo il Commodore dichiara una memoria di 64K, ma il BBC Micro 32K, l'Electron 32K, lo Spectrum 48K, il Tandy 32K e il Dragon 32K hanno tutti 64K di memoria totale. La cifra 'K' si riferisce alla quantità di memoria che si può *usare*: gli altri 32K o 16K sono riservati alla macchina. La memoria del Commodore è più flessibile, consentendo al programmatore di accedere anche alla porzione di memoria normalmente riservata a operazioni interne.

In ognuna delle macchine citate, i costruttori numerano le locazioni di memoria da ØØØØ a FFFF, ma nel più piccolo Spectrum 16K, che ha 32K di memoria in tutto, la numerazione va da ØØØØ a 7FFF.

- DI COSA È FATTA LA MEMORIA
- QUANTA MEMORIA POSSIEDE IL NOSTRO COMPUTER
- ROM E RAM
- COME È SUDDIVISA LA MEMORIA
- COSA C'È NELLA MEMORIA
- DOVE VENGONO CONSERVATI I PROGRAMMI IN **BASIC**
- COME IL COMPUTER MEMORIZZA I NUMERI MOLTO GRANDI

VOLTARE PAGINA

La memoria è organizzata in *pagine*. Ogni pagina contiene 1ØØ locazioni di memoria in hex, cioè 256 in decimale. La cosiddetta 'pagina zero' va da ØØØØ a ØØFF, la pagina uno da Ø1ØØ a Ø1FF e così via.

ROM E RAM

La memoria è di due tipi: ROM e RAM. ROM (Read-Only-Memory) vuol dire che è una memoria utilizzabile per la sola lettura delle locazioni di memoria, ma non per la scrittura. Le informazioni in essa contenute vengono fissate, con speciali apparecchi, in modo permanente. È quasi impossibile danneggiare il contenuto di una ROM (salvo spezzarla, ovviamente).

In genere, la ROM contiene il sistema operativo del computer e l'"interprete" necessario per tradurre i programmi del BASIC al codice macchina. La sua presenza in un computer impone alla restante memoria una particolare struttura, rappresentata nelle mappe di memoria riportate più sotto.

RAM (Random-Access-Memory) vuol dire memoria ad accesso casuale, ma non è così accessibile all'utente come si potrebbe pensare. Una parte di essa viene adoperata dai programmi conservati nella ROM (per la gestione del *display* e per altre funzioni specifiche). Se modifichiamo a caso i valori contenuti in queste locazioni riservate, può accadere di compromettere momentaneamente il corretto funzionamento del computer (occorrerà spegnerlo e riaccenderlo). In sostanza, la RAM è una lavagna vuota sulla quale si può scrivere e rileggere a piacimento.

I COMPITI DELLA MEMORIA

Le mappe della memoria, riportate più avanti, rappresentano graficamente il contenuto della memoria, ma senza descriverne l'esatta posizione fisica, dato che nel computer la memoria è distribuita in più *chip*. Ogni mappa illustra schematicamente lo scopo di ciascuna delle varie parti della memoria.

Solo alcune delle delimitazioni segnate tra le sezioni della mappa, come quella tra ROM e RAM, coincidono effettivamente col passaggio del chip all'altro. Altre, invece, sono flessibili e la loro esatta posizione è indicata da un *puntatore* conservato nell'area delle variabili di sistema.

Un puntatore non è che una locazione (o più precisamente due byte) di memoria nella quale viene memorizzato l'indirizzo di un'altra locazione. L'indirizzo di ogni byte di memoria è composto da due byte: ecco perché occorrono due locazioni di memoria adiacenti. Il termine "puntatore" è piuttosto azzeccato, perché viene appunto usato per *puntare* ad una particolare locazione di memoria.

I 16K di ROM dello Spectrum vanno da ØØØØ a 3FFF e contengono il programma interprete BASIC, quello per la redazione dei testi (EDITOR), varie routine di input/output e il set caratteri, composto da tutti i dati per le lettere dell'alfabeto, cifre e simboli grafici disponibili sullo Spectrum. Gli altri 48K, da 4ØØØ a 7FFF sul 16K, sono memoria RAM.

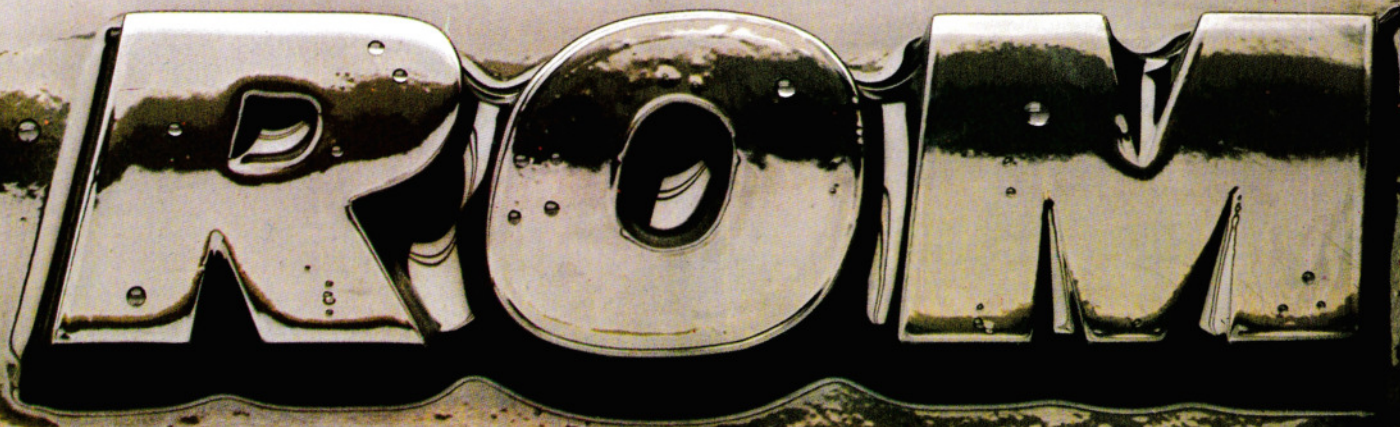
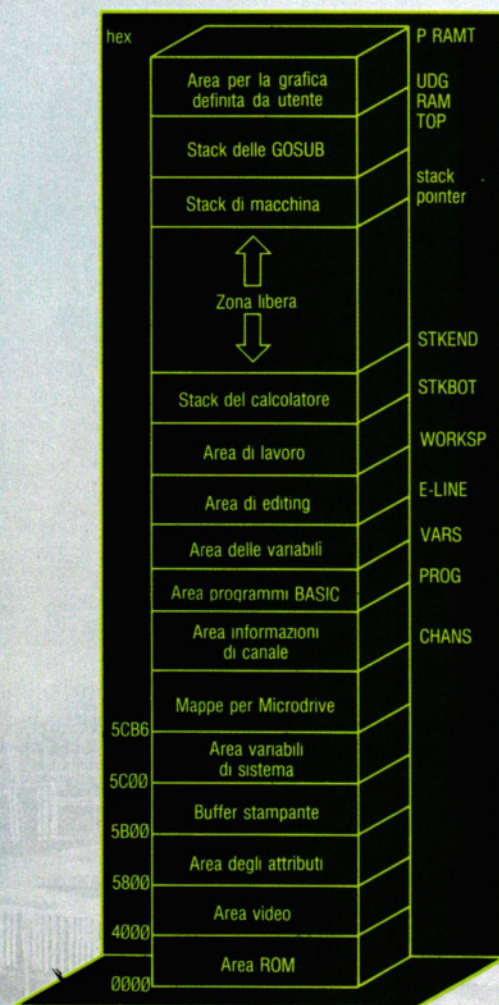
La RAM è suddivisa in zone, ognuna delle quali ha un compito specifico.

L'*area del display* controlla la visualizzazione sullo schermo TV. Ogni locazione di memoria corrisponde a una fila di otto pixel.

L'*area degli attributi* controlla i colori PAPER e INK di ciascuno dei 768 caratteri visualizzabili sullo schermo e il tipo di visualizzazione (lampeggiante o fissa, normale o BRIGHT).

Nel *buffer della stampante* vengono depositate, una alla volta, le righe di testo destinate alla stampante.

Le *variabili di sistema* sono le locazioni che conservano i puntatori e contatori usati dal sistema operativo.

L'*area della mappa dei microdrive* esi-

Mappa della memoria (Spectrum)

ste solo se si collega un Microdrive allo Spectrum. Altrimenti CHANS, il cui indirizzo è nelle locazioni 23.631 e 23.632 (5C4F e 5C50), passa a 5BC6.

È l'*area di informazioni di canale* che contiene i dati relativi all'input e all'output. Da qui, l'input da tastiera viene inviato alla parte bassa dello schermo, curando anche la visualizzazione nella parte restante dello schermo e le operazioni di stampa.

L'*area di programma BASIC* contiene le linee del programma BASIC attualmente presentate in memoria e la sua dimensione dipende dalla lunghezza del programma. L'indirizzo iniziale è dato dalla variabile di sistema PROG, contenuta alle locazioni 23.635 e 23.636 (5C53 e 5C84) nell'area delle variabili di sistema. Se il microdrive non è collegato, il puntatore vale 23.755 (5CCB in hex).

Il seguente programma esamina l'area di programma e visualizza sia il valore di ogni locazione sia, a fianco, il simbolo ASCII corrispondente. Si noti che le parole chiave del BASIC non vengono memorizzate carattere per carattere come stringhe ASCII, ma codificate in un solo byte, o al massimo due.

Questi simboli codificati (chiamati *token*), vengono decodificati e interpretati automaticamente dallo Spectrum durante l'esecuzione dei programmi BASIC.

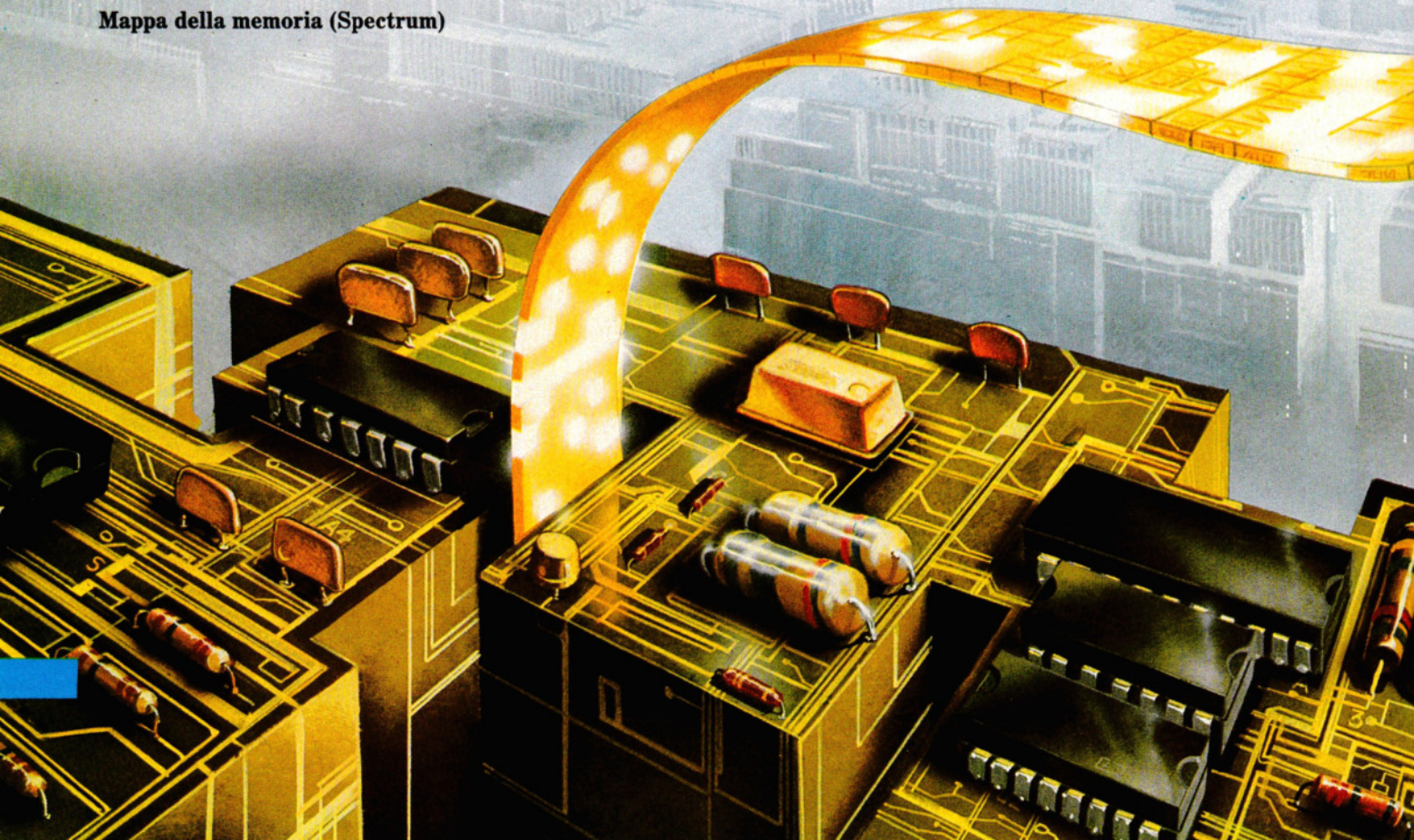
```
10 FOR n=23755 TO 23848
20 PRINT n;TAB 10;PEEK n;
30 IF PEEK n>31 THEN PRINT TAB 20;
   CHR$ PEEK n;
40 PRINT: NEXT n
```

L'*area delle variabili* memorizza i valori delle variabili adoperate dal programma BASIC in uso. Comincia da VARS, il cui indirizzo è contenuto nelle locazioni 23.627 e 23.628 (5C4B e 5C4C hex) all'interno dell'area delle variabili di sistema. Quando si esegue un RUN di un programma, l'inizio di quest'area resta sempre dov'è, ma il limite superiore cresce all'aumentare del numero e della lunghezza delle variabili adoperate dal programma.

Nell'*area di editing* viene redatta ogni linea di BASIC: per poter essere elaborata, una linea di programma deve essere ricopiata in quest'area. Dopo la pressione di ENTER essa viene codificata e trasferita nell'area del programma BASIC. L'area di redazione inizia da E-LINE, il cui indirizzo si trova nelle locazioni 23.641 e 23.642 (5C59 e 5C5A hex) nell'area delle variabili di sistema.

L'*area di lavoro* è usata dal sistema per compiti generici, quali la memorizzazione di dati immessi e la concatenazione di stringhe. WORKSP è memorizzato in 23.649 e 23.650 (5C61 e 5C62 hex) nell'area delle variabili di sistema. Quando è inutilizzato, questo spazio si riduce a zero.

Lo *stack* del calcolatore è usato per "appoggiarvi" numeri a virgola fluttuante, interi di cinque byte o serie di parametri di cinque byte, quando si tratta di strin-

ghe. Questo stack inizia da STKBOT, con indirizzo alle locazioni 23.651 e 23.652 (5C63 e 5C64 hex) e termina a STKEND, in 23.653 e 23654 (5C65 e 5C66 hex).

Dopo questa c'è un'area di memoria inizialmente vuota, che viene occupata dalla crescita dell'area di stack: se STKEND sale fino a incontrare il puntatore dello stack, lo Spectrum ci avvisa che la memoria è insufficiente (OUT OF MEMORY).

Più in alto si incontra lo *stack di macchina*, usato durante l'esecuzione di un programma BASIC. Lavorando in codice macchina lo si può manipolare direttamente, come vedremo in seguito.

Lo stack per le GOSUB contiene il numero di linea alla quale il computer deve tornare dopo aver seseguito una subroutine.

RAMTOP è il limite massimo di RAM disponibile per la scrittura di programmi: l'indirizzo è in 23.730 e 23.731 (5CB4 e 5CB5 hex) nell'area delle variabili di sistema.

Dopo ci sono 168 locazioni di memoria usate per contenere 21 caratteri UDG. Tuttavia, siccome l'indirizzo di RAMTOP si può variare, nei programmi in codice macchina si è soliti "abbassare" gli stack delle GOSUB e del calcolatore, per inserire il codice macchina. Di solito i programmi in codice macchina vengono collocati sopra RAMTOP, per evitare sovrapposizioni da parte del BASIC.
P-RAMT è il limite fisico della RAM, ossia non esistono ulteriori locazioni di memoria dopo questo indirizzo. Anche se l'indirizzo di P-RAMT è fisso, lo Spectrum lo considera una variabile di sistema, il cui valore è nelle locazioni 23.732 e 23.733 (5CB5 e 5CB6 hex). Modificando alcuni dei puntatori appena descritti, possiamo far sì che uno Spectrum da 48K si comporti come se fosse a 16K. Per verificare il valore di P-RAMT si digiti:

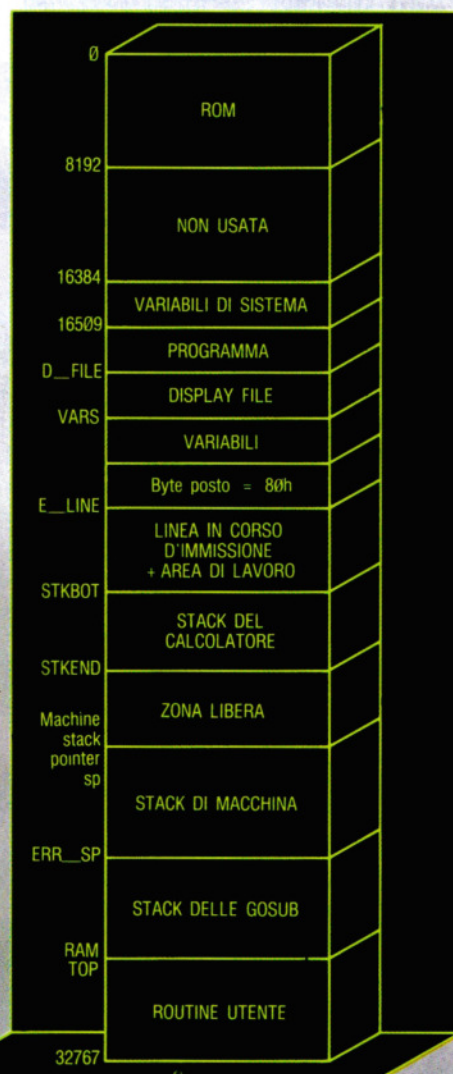
```
PRINT PEEK 23732+256*PEEK 23733
```

Si ottiene 65535 sullo Spectrum 48K e 32767 sul modello a 16K. Se non si sono modificati i puntatori e i valori non corrispondono, c'è qualcosa che non va nella memoria dell'apparecchio. Ogni puntatore delle variabili di sistema può essere esaminato con una simile linea PRINT, sostituendo opportunamente i valori delle due locazioni della variabile. L'istruzione PEEK del BASIC legge il contenuto del byte di memoria specificato e ne visualizza l'equivalente decimale sullo schermo.

Essendo un indirizzo lungo due byte, richiede due locazioni di memoria. Lo Spectrum separa gli indirizzi di 4 cifre hex, ad esempio l'indirizzo normale RAMTOP FF57, in due parti: FF e 57.

Il byte meno significativo (o "basso"), 57, va nella locazione più bassa e il byte più alto FF, va in quella più alta. Può sembrare un metodo strano, ma allo Spectrum torna più semplice così. Ecco perché nel programma abbiamo moltiplicato per 256 la seconda locazione di memoria: senza farlo, la PEEK fornirebbe l'equivalente decimale FF invece di FF00.

Gli 8K di ROM dello ZX81 vanno da 0000 a 1FFF in hex, da 0 a 8.190 in decimale. Gli altri 8K, da 2000 a 3FFF, non sono utilizzati. La mappa della memoria qui sotto si riferisce ad uno ZX81 dotato di una espansione di 16K di RAM, che va da 4000 a 7FFF in hex o da 16.384 a 32.767 in decimale. Se si possiede l'espansione di 8K o di soli 4K di RAM, tutte le aree di memoria qui mostrate sono semplicemen-

Mappa della memoria (ZX81)

te compresse in 8K o 4K portando il limite fisico della RAM A 5FFF, 24.575 decimale, o 4FFF, 20.479 decimale.

Ovviamente, se lo ZX81 non è dotato di alcuna espansione, tutto quanto è compresso in 1K: da 4ØØØ a 43FF hex, da 16.384 a 17.407 decimale. In tutti questi casi le *variabili di sistema* occupano l'area da 4ØØØ a 4Ø87 hex, da 16.384 a 16.509 decimale. Le altre aree non sono fisse e i particolari dei loro confini (D-FILE, VARS, E-LINE, STKBOT, STKEND, ERR-SP, RAMTOP) sono memorizzati come puntatori nella stessa area delle variabili di sistema.

La prima di queste aree flessibili, da 16.509 a D-FILE, continene il programma BASIC in corso di redazione: è dunque possibile scriverne uno che "legga se stesso", esaminando quest'area. È proprio ciò che fa il seguente programma: esamina quest'area e visualizza il valore di ciascuna locazione. A fianco, fa comparire il simbolo ASCII corrispondente al valore, onde riconoscere i caratteri leggibili. Le parole chiave del BASIC non sono memorizzate carattere per carattere come stringhe ASCII, ma codificate in un solo byte (*token*), che viene in seguito decodificato e interpretato da una complessa routine. Questo semplice programma non è in grado di fare la conversione:

```
10 FOR n=16509 TO 16704
20 PRINT n;TAB 10;PEEK n;
30 IF PEEK n>31 THEN PRINT TAB 20;
   CHR$ PEEK n
40 PRINT
50 NEXT n
```

Da D-FILE a VARS c'è la zona di memoria dove è conservata l'immagine sullo schermo (*display file*).

L'area delle variabili si espande quando, eseguendo un programma, vengono definite nuove variabili. Essa termina con una locazione contenente un 80 in hex.

Tra E-LINE e STKBOT ci sono l'*area di editing* e l'*area di lavoro*. In quest'ultima viene elaborata la linea in corso di redazione, prima di essere aggiunta al resto del programma. Premendo NEW LINE, la linea viene trasferita nell'area del programma.

Quest'area ha due ruoli: così come l'area di lavoro viene usata sia per contenere dati in ingresso che per concatenare stringhe.

I numeri usati quando si fanno operazioni aritmetiche in BASIC sono depositati nello *stack del calcolatore* tra STKBOT e STKEND.

Da STKEND al puntatore dello stack della macchina c'è un'area di memoria vuota nella quale le due aree a fianco possono estendersi. Lo *stack di macchina* è utilizzato durante l'esecuzione dei programmi BASIC, mentre lo *stack delle GOSUB* è usato per memorizzare il numero di linea alla quale lo ZX81 deve tornare dopo una subroutine. Il suo funzionamento è argomento di una lezione successiva.

Normalmente RAMTOP è il llimite fisico della memoria, cioè 32.767, 24.575, 20.479 o 17.407, a seconda dell'espansione RAM collegata. Lo si può conoscere leggendo il puntatore pertinenete nell'area delle variabili di sistema, con una linea di questo tipo:

```
PRINT PEEK 16388+256*PEEK 16389
```

Questa visualizza il valore del limite di memoria del proprio apparecchio.

Si può anche depositare in questi puntatori un valore tale da abbassare con RAMTOP il limite fisico della memoria, lasciando così un'area protetta per le routine in codice macchina, dove non sovrapposte dal BASIC. Tuttavia, una volta scritto in quest'area, il codice macchina non può venire salvato (SAVE) su nastro.

Infati, l'unica area di memoria che si può trasferire su nastro è l'area BASIC: come depositare i programmi in codice macchina in quest'area mantenendoli protetti sarà argomento di una successiva lezione.

La memoria del Commodore ha una natura diversa da quella degli altri home computer. Tanto per cominciare, non c'è una distinzione rigida tra ROM e RAM, per cui alcune parti della memoria possono indifferentemente appartenere a una delle due zone.

I chip della memoria del Commodore offrono di fatto 64K di RAM, ma, poiché il massimo indirizzo gestibile ariva solo a 65.535 o FFFF in hex, non ci sarebbero altri indirizzi da assegnaare ai chip della ROM. La soluzione è stata quella di assegnare alla ROM alcuni indirizzi, comuni anche alla RAM. Per stabilire se leggere ROM o RAM, il computer esamina i bit contenuti nella locazione uno.

Una zona di memoria appartiene alla ROM se il corrispondente bit vale 1, altrimenti si tratta di RAM.

Nei capitoli precedenti, si è incontrata l'istruzione POKE, seguita sempre da due numeri, di solito in decimale. Il primo è compreso tra Ø e 65.535 e costituisce l'indirizzo di memoria. Il secondo è minore di 255, il massimo numero decimale che una sola locazione possa contenere: POKE scrive il secondo numero nella locazione individuata dal primo. Si ricordi che la scrittura può avvenire soltanto in locazioni RAM: la ROM, ricordiamo, è la memoria a sola lettura. Se si scrive un numero in un indirizzo condiviso sia da ROM che da RAM, il Commodore deve necessariamente depositarlo nella RAM. L'istruzione BASIC complementare a POKE è PEEK, che legge il contenuto della locazione specificata. La lettura può riguardare sia locazioni ROM che RAM, quindi il computer deve vedere se il bit è Ø per RAM o 1 per ROM.

All'accensione, gli appropriati bit vengoo tutti posti a 1, abilitando soltanto la ROM.

Le 8.192 locazioni di memoria in cima (2ØØØ in hex), da EØØØ a FFFF, contengono il *Kernal su ROM*, cioè le istruzioni fondamentali della macchina. Nondimeno, anche questa ROM può esere disabilitata, liberando 8K di RAM. Il Commodore, però, non risponderà più ai comandi, a meno che non si immetta un altro sistema operativo o non si ricopi, anche modificato, quello del Kernal.

Il blocco subito sotto, da DØØØ a DFFF, è più complesso. Qui si possono trovare i dispositivi ROM di input/output oppure il set di caratteri ROM o, ancora, 4K di RAM. Normalmente, sono selezionati i dispositivi di input/output, ma quando è necessario lavorare sul set dei caratteri, la parte richiesta viene fedelmente riprodotta nella RAM riservata al BASIC (da 1ØØØ a 1FFF). Sia il set dei caratteri che le ROM di input/output possono essere deselezionate, liberando 4K di RAM, ma si impedirebbe ogni immissione da tastiera o emissione su schermo.

Il successivo blocco di 4K di memoria, da CØØØ a CFFF, è di RAM, normalmente impiegata per i programmi in codice macchina. Da AØØØ a BFFF risiede l'interprete BASIC: anche questa ROM può essere "spenta" per utilizzare un altro linguaggio o modificare il BASIC ricopiandolo nella RAM. Più sotto ci sono i 26K della RAM per i programmi BASIC, di cui una parte è riservata a speciali funzioni. Gli 8K, da 8ØØØ a 9FFF, sono dedicati alle cartucce e gli 8K da 2ØØØ a 4ØØØ sono usati per lo schermo ad alta risoluzione. L'area da Ø4ØØ a Ø7E7 serve per lo schermo testuale, mentre quella da Ø7F8 a Ø7FF contiene i puntatori ai dati degli sprite. Nel modo ad alta risoluzione, quest'area serve anche per la tabella dei colori. Questo programma esamina la RAM dedicata ai programmi BASIC:

```
10 FOR N=2048 TO 2143
20 PRINT N;TAB(10);PEEK(N);
```

```
30 IF PEEK(N)>31 THEN PRINT TAB(20);
   CHR$(PEEK(N)):NEXT N:END
40 PRINT:NEXT N
```

In pratica, il programma esamina se stesso, visualizzando gli equivalenti decimali dei valori contenuti in ciascuna locazione assieme ai corrispondenti caratteri ASCII. Il computer visualizza variabili, valori numerici, segni matematici e punteggiatura nel programma, ma non le parole chiave.

Queste, infatti, non sono memorizzate come stringhe ASCII, ma come numeri di un solo byte, talvolta due, detti *token*.

La memoria da Ø4ØØ a ØØØØ contiene, l'*area di lavoro* e le *variabili di sistema*, di cui una, la locazione uno, contiene i bit che controllano se le ROM sono abilitate o no. Si può leggere questo byte con:

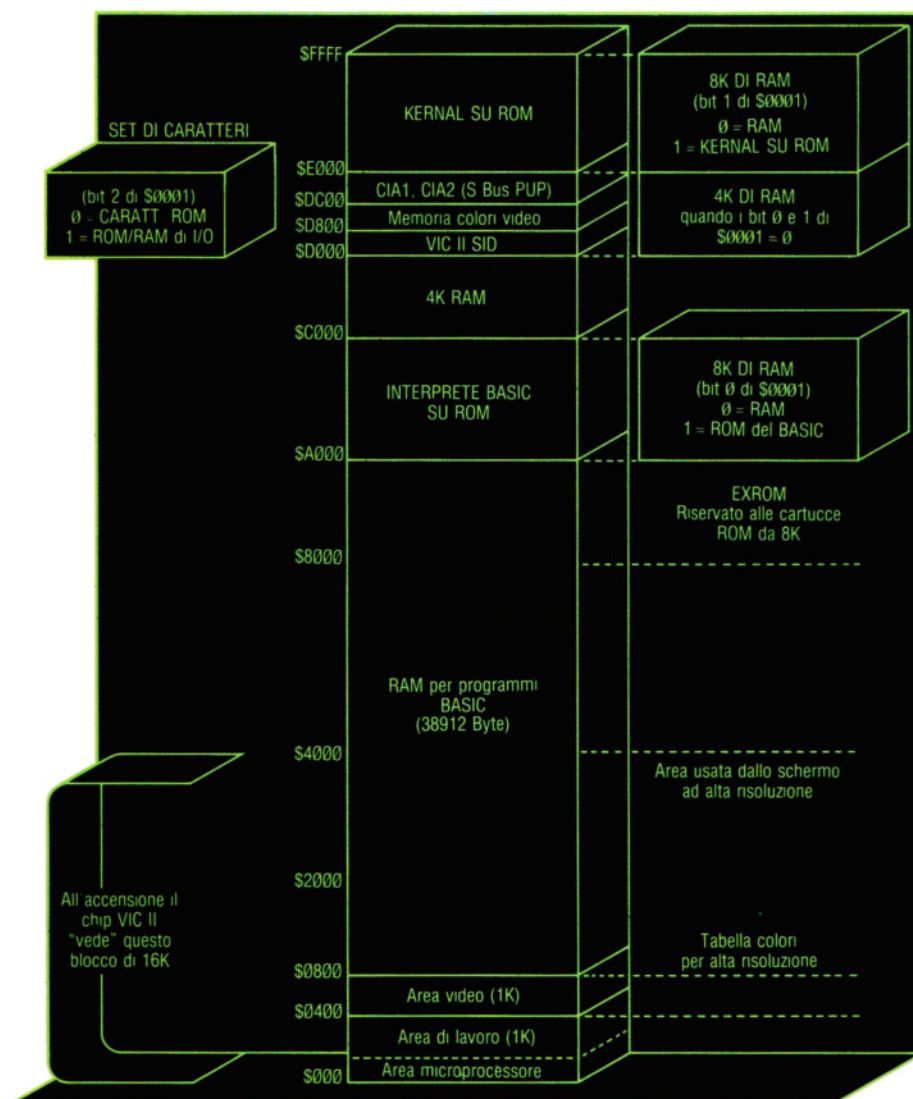
```
PEEK(1)
```

Normalmente, viene visualizzato il valore decimale 55: convertendo il valore in binario, si conosce la configurazione della ROM.

Se il bit zero (cioè quello all'estremità destra, dato che tutto si numera da zero nel computer) vale zero, significa che la ROM del BASIC è disabilitata. Se il bit uno, quello accanto a sinistra, vale Ø, allora la ROM Kernal è disabilitata a vantaggio del set dei caratteri. Per spegnere sia i dispositivi di input/output che il set dei caratteri per liberare gli altri 4K di RAM, occorre spegnere la ROM Kernal e la ROM del BASIC.

I bit da tre a cinque controllano un registratore a cassette e i bit sei e sette non sono utilizzabili.

Il decimale 55 vale ØØ11Ø111 in binario, quindi i bit zero, uno e due sono tutti posti a 1 e tutte le ROM sono abilitate.

Mappa della memoria (Commodore 64)

Il chip 65Ø2 del Vic 2Ø accede a 64K locazioni di memoria, ma il Vic di base usa in tutto solo 29K di memoria. Si posono però aggiungere espansioni di ROM e di RAM fino a ottenere 35K.

La mappa della memoria a pagina 214 mostra la struttura della memoria e dove trovano posto le espansioni per il Vic 2Ø.

Le *variabili di sistema* occupano le prime 1.Ø24 locaizoni, da ØØØØ a ØØ3FF in hex o da Ø a 1.Ø23 in decimale. Quest'area contiene speciali puntatori, che segnano i vari confini nel resto della RAM e danno alla memoria del Vic una struttura complessiva. La configurazione della memoria si può modificare, depositando in queste locazioni valori diversi.

La *memoria utente* va da Ø4ØØ a 7FFF, da 1Ø24 a 32.767 in decimale: in tutto 31K di memoria, molta della quale ottenuta con le espansioni RAM. I primi 7K, da Ø4ØØØ 1FFF hex, da 1.Ø24 a 8.191 decimale, sono esclusivamente di RAM, ottenuta con un'espansione di 3K (da Ø4ØØ a ØFFF hex, da 1.Ø24 a 4.Ø95 decimale) e con 4K di RAM di sistema (da 1ØØØ a 1FFF hex, da 4.Ø96 a 8.191 in decimale).

La maggior parte di quest'area (da ØFFF a 1DFF hex, da 4.Ø96 a 7.679 decimale), è riservata ai programmi BASIC digitati da tastiera. Il seguente programma BASIC è in grado di leggere quest'area, permettendoci di esaminare il valore contenuto in ciascuna locazione, assieme al carattere ASCII corrispondente. Vedremo i nomi delle variabili, i numeri, i segni aritmetici e le punteggiature, ma non le parole chiave del BASIC. Queste infatti non sono memorizzate come stringhe ASCII, ma codificate in un solo byte, a volte due, chiamati *token*.

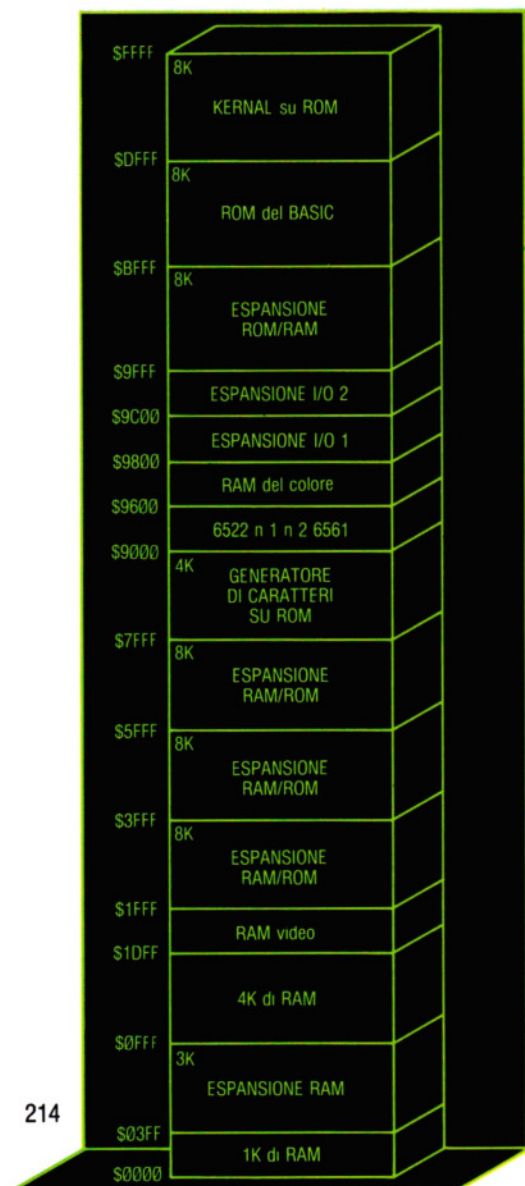
```
10 FOR N=4096 TO 4191
20 PRINT N;TAB(10);PEEK(N)
30 IF PEEK(N)>31 THEN PRINT TAB(20);
   CHR$(PEEK(N)):NEXT N
40 PRINT:NEXT N
```

In assenza di altre espansioni, le ultime 5212 locazioni di memoria (da 1FØØ a 1FFF hex, da 7.68Ø a 8.191 decimale), costituiscono la *RAM dello schermo*, dove si conserva l'immagine video, altrimenti questa slitta a 1ØØØ hex, 4.Ø96 in decimale. Le restanti tre sezioni da 8K di memoria utente possono essere RAM o ROM, a seconda del tipo d'espansione innestata e sono tutti a disposizione dell'utente, eccetto un 1/2K di memoria per lo schermo.

Il *generatore di caratteri* va da 8ØØØØ a 8FFF hex, da 32.768 a 36.863 decimale: è un'area ROM di 4K contenente le confi-

gurazioni di pixel per ognuno dei 255 caratteri ASCII visualizzabili sullo schermo. Le *interfacce di input/output e di controllo del sistema* occupano da 9ØØØ a 912F hex, da 36.864 a 37.167 decimale. Quest'area comunica direttamente con i tre chip di I/O (due chip 6522 VIA e il chip 5621 VIC), controllabili mediante specifiche POKE. Se non ci sono espansioni, le locazioni di memoria da 94ØØ a 95FF hex, da 37.888 a 38.399 decimale, riguardano la *memoria colore*.

Ognuno dei 5Ø6 byte in questo blocco fissa il colore del primo piano e dello sfondo del corrispondente byte nella memoria video. Con più di 7K di RAM per l'utente, l'inizio della memoria colore si innalza a 96ØØ hex, 38.4ØØ decimale.

L'espansione di 8K da AØØØ a BFFF hex, da 4Ø.96Ø a 49.151 decimale, può servire per programmi memorizzati nella ROM.

All'accensione, il sistema operativo può far partire programmi in codice macchina situati in questa zona, anziché del BASIC.

L'*interprete BASIC*, che occupa da CØØØ fino a DFFF hex, da 49.152 a 57.343 decimale, traduce i programmi dal BASIC al codice macchina durante l'esecuzione. Il Kernal su ROM, da EØØØ a FFFF hex, da 57.344 a 69.535 decimale, contiene il *sistema operativo* che controlla il funzionamento della macchina. È questo che stabilisce la configurazione di base della RAM, all'accensionedell'apparecchio, determinando le demarcazioni tra le aree, che esistono in quanto specificate da questo settore specializzato della ROM.

Mappa della memoria (Vic 2Ø)

Nel BBC Micro e nell'Electron, la ROM occupa le locazioni di memoria da &FFFF a &8ØØØ.

Quest'area contiene il sistema operativo, l'interprete BASIC e una zona che si occupa di input e output alle periferiche, come stampanti, registratori a cassette e simili.

L'area da &7FF a HIMEM è riservata alla *memoria video*, la cui estensione dipende dal MODE usato. HIMEM scende fino a &3ØØØ per i MODE Ø, 1 e 2; a &4ØØØ per il MODE 3, a &5ØØØ per i MODE 4 e 5; a &6ØØØ per il MODE 6 e (solo per il BBC) a &7ØØØ per il MODE 7.

La zona tra HIMEM e LOMEM è l'*area di lavoro* usata per il programma BASIC in esecuzione. In genere, LOMEM e TOP (puntatori situati nell'area delle variabili di sistema) puntano alla stessa zona, ossia il primo indirizzo libero dopo la fine del programma BASIC. Si possono però separare scrivendo un nuovo indirizzo nelle locazioni relative a LOMEM: l'area in mezzo può adesso ospitare un programma in codice macchina.

I programmi in BASIC occupano l'area tra TOP e PAGE, che normalmente vale &EØØ. Anche il puntatore PAGE può essere modificato, per ricavare lo spazio per un programma in codice macchina. Si può osservare cosa succede nell'area dei programmi BASIC utilizzando questo programma:

```
10 FOR N=PAGE TO TOP
20 PRINT;~N;TAB(15);?N;
30 IF ?N>31 THEN PRINT
   TAB(30);CHR$(?N);
40 PRINT
50 NEXT
```

Esso visualizza l'indirizzo esaminato, in valore hex in esso contenuto e il carattere corrispondente. Tuttavia, non apparirà alcuna parola chiave del BASIC, dal momento che queste non sono memorizzate sotto forma di stringa di caratteri, ma codificate in un valore di un solo byte, talvolta due, chiamato *token*.

La pagina 13 (tra &DØØ e &DFF), può essere usata per brevi programmi in codice macchina, purché non si adottino unità a dischetti, alle quali essa è riservata.

Mappa della memoria (Acorn)

La pagina 12 (da &6ØØ s &CFF), è riservato alla grafica definita dall'utente e la pagina 11, da &BØØ a &BFF, contiene le definizioni date ai tasti programmabili dall'utente.

La pagina 1Ø, da &AØØ e &AFF, è il *buffer di input*, usato per il trasferimento di dati provenienti dal registratore a cassette oppure da un'interfaccia collegata a un altro computer, prima di inviarli alla parte di memoria competente.

La pagina 9, da &9ØØ a &9FF, è il *buffer dell'output*, che ha una funzione analoga al buffer di input, ma in uscita dal sistema.

La pagina 8, da &8ØØ a &8FF, contiene l'*area di lavoro sul suono*, dove sono sintetizzati i vari suoni prodotti dal computer. Essa contine anche il *buffer del suono* e il *buffer della stampante* nei quali vengono "appoggiati" i suoni e i dati, prima di essere emessi.

Le pagine 7, 6, 5 e 4 sono utilizzate dall'interprete BASIC nella ROM per tradurre i programmi in un linguaggio accessibile al computer.

La pagina 3, da &3ØØ a &3FF, contiene il *buffer della tastiera*, parti attinenti al cursore del testo e vari parametri grafici.

La pagina 2, da &2ØØ a &2FF, contiene i parametri operativi, che, cambiando continuamente durante il funzionamento della macchina, non possono essere memorizzati nella ROM, che è memoria a sola lettura.

La pagina 1, da &1ØØ a &1FF, è lo *stack della macchina*, un'area di memoria, utilizzata dal computer durante l'esecuzione di programmi BASIC. All'occorrenza, si può intervenire su di essa, per inserire programmi in codice macchina.

La *pagina zero*, da &Ø a &FF, è dedicata a molte funzioni speciali nel BBC e nell'Electron, dal momento che vi si accede con facilità e che i suoi indirizzi occupano un solo byte: il secondo byte (quello più significativo) è sempre zero (ØØ).

Dragon e Tandy hanno una ROM di 32K e un'area di input/output da &H8ØØØ a &HFFFF, che controlla, oltre allo scambio di dati tra il computer e le cartucce aggiuntive, anche il sistema operativo e l'interprete BASIC. Tra &H36ØØ e &H7FFF, sono memorizzati i programmi in BASIC e le relative variabili.

In questa stessa area vengono eseguiti i programmi, a seguito di un RUN. Lo *stack* (un'area specializzata di memoria di cui si parlerà più avanti) occcupa la parte più alta della RAM, collocata generalmente subito sotto la ROM (da &H7FFF in giù). Lo stack può essere abbassato, per far posto a un programma in codice macchina, cambiando il valore della *variabile di sistema* che stabilisce l'indirizzo di base dello stack: questa procedura viene spiegata più avanti.

Il seguente programma esamina l'area riservata ai programmi BASIC: legge e visualizza i valori hex contenuti nelle varie locazioni di memoria che contengono il programma. A fianco compaiono in caratteri ASCII equivalenti.

Potremo riconoscere le variabili, i valori numerici, i segni aritmetici e la punteggiatura del programma, ma non le parole chiave del BASIC. Queste, infatti, sono codificate in valori di un solo byte, talvolta due, detti *token*. Questo programma è troppo semplice per decodificare i token (occorre una speciale routine contenuta in ROM e alcune variabili di sistema), perciò al loro posto visualizza blocchi grafici.

```
10 PCLEAR4
20 FOR N=7681 TO 7744
30 PRINT N,PEEK(N);"□";CHR$(PEEK(N))
40 NEXT
```

Entrambi i computer possiedono otto pagine grafiche ad alta risoluzione, da non confondersi con le pagine di memoria (citate a pagina 2Ø9). Ciascuna di quelle è composta da 256 locazioni, ossia 1/4K. Le pagine grafiche, invece, sono lunghe 1/2 K e ognuna contiene sufficienti informazioni grafiche per riempire lo schermo nel modo a più bassa risoluzione, PMODEØ, mentre per quello a più alta risoluzione, PMODE4, occorrono quattro pagine per videata. Il computer riserva automaticamente quattro pagine, ma con PCLEAR il numero può aumentare. Si posson quindi memorizzare da una a otto videate di grafica in memoria.

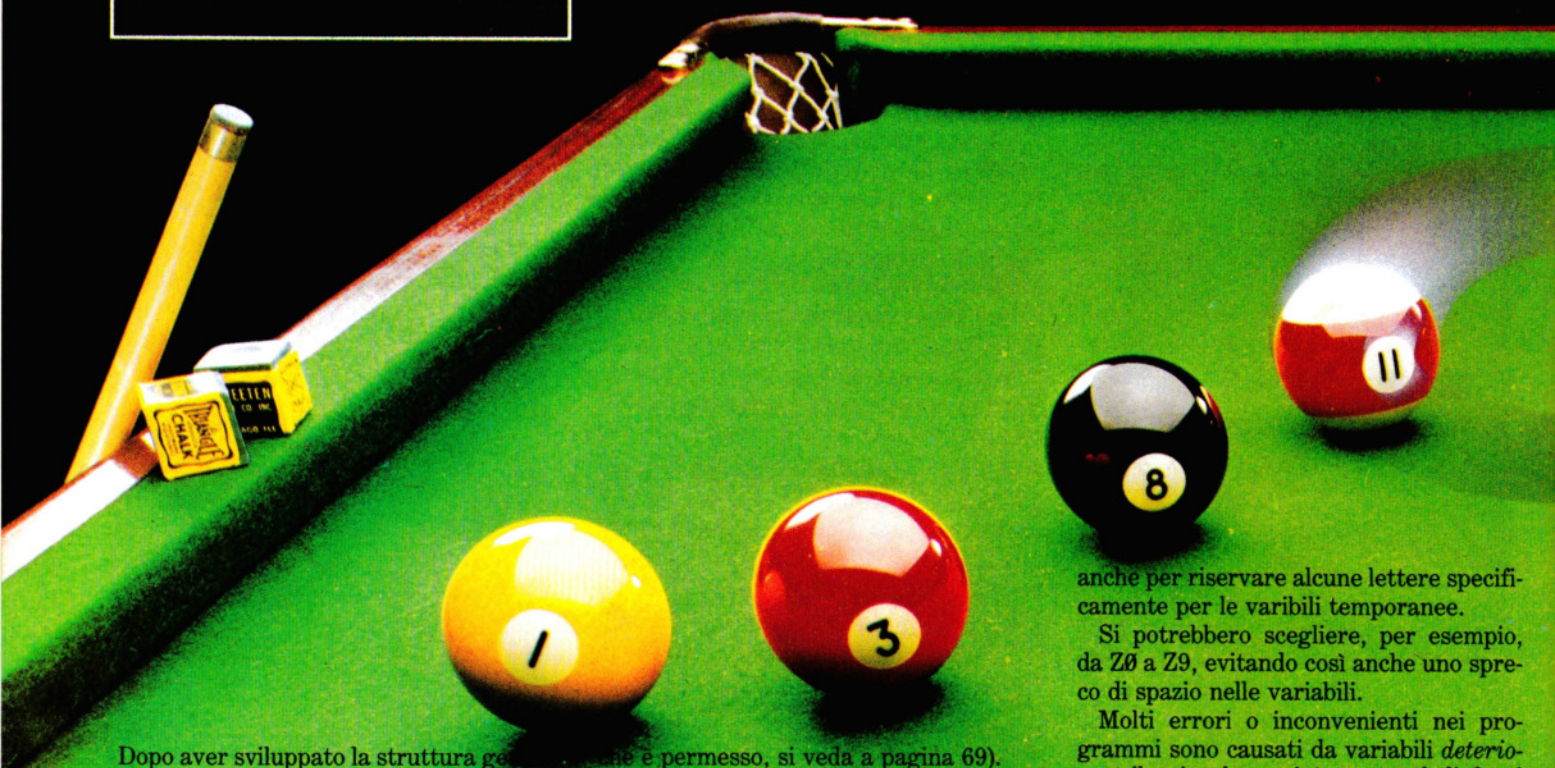
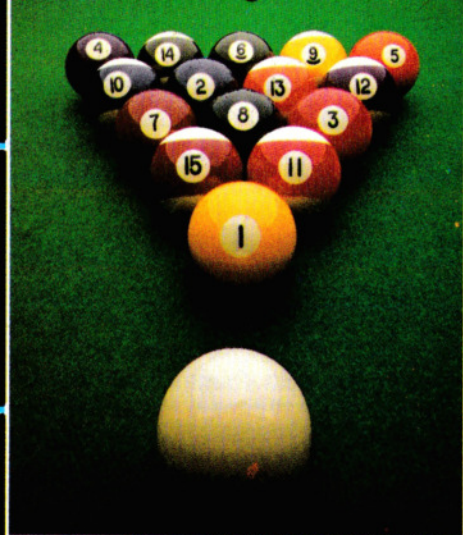
Se non utilizzate, le pagine grafiche sono vuote e, se serve memoria, possono venir ridotte dal BASIC, fino a pagina uno, ottenendo dieci 1/2 K di memoria in più per i propri programmi.

Lo *schermo testuale* occupa 1/2 K tra &H4ØØ e &H6ØØ e ha una locazione per ogni posizione di carattere sullo schermo, che ha 16 righe di 32 caratteri ciascuna. I caratteri, formati su una griglia di 96 pixel (otto per dodici pixel), sono creati da un *chip generatore video* a sé, che non fa parte della memoria.

Da &HØØØ a &H4ØØ ci sono le *variabili di sistema*, un insieme di puntatori che forniscono gli indirizzi iniziali di varie aree o di altre variabili di sistema.

Il primo gruppo di 256 byte, da &HØØ a &HFF, è conosciuto come *pagina diretta*. Essa differisce dalla pagina zero, dal momento che qualsiasi pagina può essere scelta per diventare una pagina diretta e contenere poi indirizzi di locazioni di un byte invece dei consueti due. Per fare ciò, occorre predisporre uno speciale *registro della pagina diretta* nel microprocessore, come vedremo più avanti. Da quanto detto, se si lasciano seguire al computer le proprie procedure, è evidente che non rimane spazio libero per programmi in codice macchina: come ricavare questo spazio, sarà detto in una prossima lezione.

Mappa della memoria (Dragon/Tandy)

UN PO' DI FORMA NEI PROGRAMMI

Dopo aver sviluppato la struttura generale del programma e scritti i singoli moduli, si può passare alla verifica di questi e infine a combinarli insieme.

In genere, le subroutine (o moduli) devono collegarsi in qualche modo al resto del programma e ciò avviene tramite variabili.

Tali variabili vengono chiamate *parametri di input o parametri di output*, a seconda che vengano passate dal programma principale alla subroutine o viceversa. È molto importante che le variabili siano specificate in modo preciso, per non generare confusione o errori.

All'inizio di un programma conviene inserire un elenco di tutte le variabili impiegate, descrivendone l'uso e, eventualmente, anche il possibile valore. Altrimenti si rischia, riprendendo il programma in seguito, di non ricordarsi più l'uso e lo scopo di gran parte delle variabili. Se il BASIC e la quantità di memoria disponibile lo consentono, si adoperino nomi lunghi per le variabili (per un quadro di ciò che è permesso, si veda a pagina 69).

Ecco come si potrebbero specificare le variabili per una routine di ordinamento:

Routine "Bubble sort"
Riordinare alfabeticamente parte o tutti gli elementi di una matrice.

Variabili immesse:

A $ (N)	matrice a una dimensione da ordinare (grandezza di N ⟩= 1)
N1	prima voce da ordinare nella matrice (1⟨ = N1 ⟨ =N2)
N2	ultima voce da ordinare nella matrice (N1 ⟨= N2⟨= dimensione di A)

Variabili in output:

A$ (N) matrice ordinaria

Variabili temporanee:

Z, Z$, I

È utile annotare le variabili temporanee usate in una subroutine, per evitare contraddizioni con il resto del programma e anche per riservare alcune lettere specificamente per le varibili temporanee.

Si potrebbero scegliere, per esempio, da ZØ a Z9, evitando così anche uno spreco di spazio nelle variabili.

Molti errori o inconvenienti nei programmi sono causati da variabili *deteriorate* il cui valore, cioè, muta al di fuori dalle previsioni. Ciò accade se, inavvertitamente, adoperiamo una variabile per due scopi diversi.

L'elenco delle variabili torna utile soprattutto se, in un secondo momento, occorre apportare modifiche al programma: gli interventi saranno più rapidi e si evita il rischio di introdurre errori deteriorando variabili già usate per altri scopi.

Ovviamente, anche i cambiamenti operati devono essere annotati. Ed ecco il programma per la routine di ordinamento o *bubble sort*. La spiegazione e il diagramma a blocchi sono a pagina 219.

```
1000 REM BUBBLE SORT (A$(N), N1, N2)
1010 LET Z=0
1020 FOR I=N1 TO N2-1
1030 IF A$(I)<=A$(I+1) THEN GOTO
   1080
1040 LET Z$=A$(I)
1050 LET A$(I)=A$(I+1)
1060 LET A$(I+1)=Z$
1070 LET Z=1
```

Andiamo ancora avanti con la programmazione strutturata, analizzando come sviluppare un programma "bubble sort" in grado di ordinare dati di qualsiasi genere

- PIANIFICARE L'USO DELLE VARIABILI NEI PROGRAMMI
- SCRIVERE LE SUBROUTINE
- COME PROVARE I MODULI
- METTERE TUTTO INSIEME

```
1080 NEXT I
1090 IF Z=1 THEN GOTO 1010
1100 RETURN
```

Dato che questa è una subroutine, andrà richiamata dal programma principale. Per esempio, la routine per ordinare gli elementi da 5 a 20, può venire richiamata così:

```
100 LET N1=5: LET N2=20:GOSUB
    1000:REM bubble sort
```

Occorrerà anche una sezione di programma per l'immissione degli elementi da ordinare e un'altra che mostri la lista ordinata. È a questo livello che va scelta la forma per la visualizzazione sul video, eventualmente disegnandola su un foglio.

PROCEDURE E FUNZIONI

Il BASIC del BBC, rispetto ad altri, ha un ottimo metodo per richiamare e definire le subroutine: mediante la definizione di PROCedure e funzioni, contraddistinte dal loro nome, anziché dal loro numero di linea. Di conseguenza, l'istruzione GOSUB, sebbene disponibile, viene raramente usata sugli Acorn.

La subroutine di ordinamento può essere scritta così:

```
1000 DEF PROCbubblesort(N1,N2)
1002 LOCAL NOSWAP, Z$,I
1004 REPEAT
1010 NOSWAP=TRUE
1020 FOR I=N1 TO N2-1
1030 IF A$(I)<=A$(I+1) THEN GOTO
     1080
1040 Z$=A$(I)
1050 A$(I)=A$(I+1)
1060 A$(I+1)=Z$
1070 NOSWAP=FALSE
1080 NEXT I
1090 UNTIL NOSWAP
1100 ENDPROC
```

Per ordinare elementi da 5 a 20 potrebbe venire richiamata in questo modo:

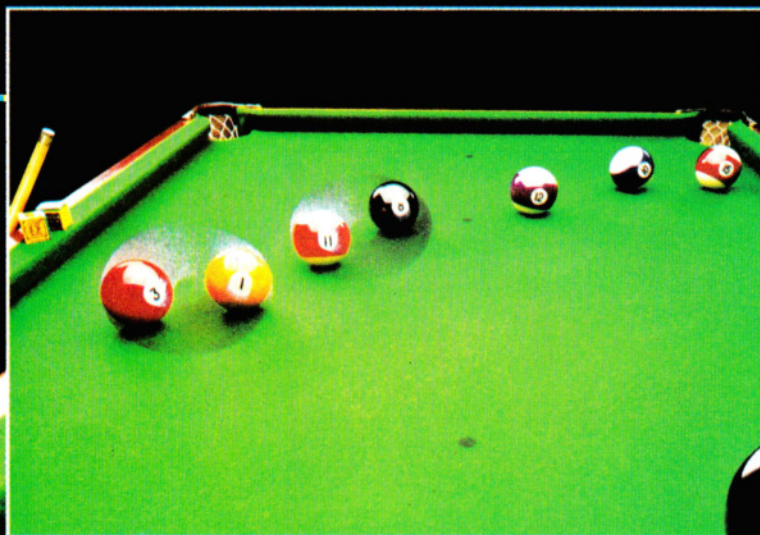
```
100 PROCbubblesort(5,20)
```

Si noti come i parametri di input, 5 e 20, si possonoi specificare assieme all'istruzione di chiamata PROC.

Si noti anche come le variabili temporanee sono dichiarate LOCAL all'interno della procedura. È questo un modo di definire una nuova variabile da usarsi limitatamente alla stessa routine, cessando di esistere al termine di questa. L'uso di variabili *locali* evita la possibilità di deteriorare quelle *globali*, usate cioè nel programma principale.Non importa più se il nome dei due tipi di variabile è lo stesso, tanto vengono tenute separate. le procedure sostituiscono qualsiasi subroutine richiamabile con la GOSUB, ma se fosse prevista una sola variabile, come risultato, può essere conveniente usare invece una funzione. Non è il caso del bubble sort (in cui è prevista l'immissione di un'intera lista di variabili), ma ecco un tipico esempio di funzione. Questa fornisce il giorno della settimana (da 1 a 7) in base alla data:

```
1000 DEF FNgiornosett(GIORNO,MESE,ANNO)
1010 LOCAL M,A
1020 IF MESE>2 THEN GOTO 1060
1030 M=MESE+10
1040 A=ANNO-1
1050 GOTO 1080
1060 M=MESE-2
1070 A=ANNO
1080 GIORNO=(((26*M-2)DIV10)+GIORNO
     +6+A+(A DIV4)+1)MOD7+1
```

Per usare questa funzione, occorre qualche altra linea per immettere una data e per visualizzare il risultato:

```
100 INPUT GIORNO,MESE,ANNO
200 PRINT FNgiornosett(GIORNO,MESE,ANNO)
300 END
```

Si noti che l'anno deve essere immesso come 84, per esempio, e non come 1984. La forma in cui si visualizzano i risultati non è la migliore, ma questa è solo un semplice esempio, perfezionabile.

PROVA DEI MODULI

Ogni modulo del progetto originario diviene, alla fine, una subroutine del programma. L'utilità di questa suddivisione si apprezza durante la fase di prova, dato che ogni modulo può venire saggiato e corretto individualmente. L'idea consiste nel definire le variabili da immettere, richiamare la subroutine e controllare i risultati. La routine di ordinamento può venire provata così:

```
8 INPUT "NUMERO DI ELEMENTI□";N
10 DIM A$(N)
12 PRINT "IMMETTERE GLI ELEMENTI"
14 FOR I=1 TO N: INPUT A$(I): NEXT I
16 INPUT "ARCO DI RIORDINO";N1,N2
18 GOSUB 1000
20 PRINT "ELENCO RIORDINATO:"
22 FOR I=1 TO N: PRINT A$(I): NEXT I
24 GOTO 16
```

Il programma gira sul Dragon, ma con un comando supplementare per liberare spazio in memoria:

```
6 CLEAR 1000
```

Sullo Spectrum si cambi la linea 10 in:

```
10 DIM A$(N,10)
```

Sullo ZX81 le lineee con istruzioni multiple vanno separate e la linea 8 cambia in:

```
8 PRINT "NUMERO DI ELEMENTI"
9 INPUT N
```

Sugli Acorn cambiare i punti e virgola alle linee 8 e 16 in una virgola e la 18 in:

```
18 PROCbubblesort(N1,N2)
```

Nei programmi molto complessi è impossibile provare ogni singolo caso di imput e output: ci vorrebbe troppo tempo. Si possono però esaminare alcuni casi limite, per verificare se il programma li individua correttamente. Per esempio, ecco una routine che andrebbe controllata immettendo i valori Ø, 1, 99 e 1ØØ.

```
1010 INPUT "IMMETTERE UN NUMERO (1
     -99)";N
1020 IF N<1 OR N>99 THEN GOTO 1010
1030 RETURN
```

(Sugli Acorn si usi una virgola invece del punto e virgola alla linea 1Ø1Ø.) È una buona norma verificare ogni linea del programma eseguendola almeno una volta durante la fase di correzione. Si dovrebbero altrettanto verificare tutte le diramazioni (salti condizionati e non), provando sia la condizione vera che la falsa.

METTERE TUTTO INSIEME

Alla fine, tutti i moduli devono essere collegati e il programma provato nell'insieme: questa è la cosiddetta *integrazione* del programma. Se le tappe precedenti sono state svolte con cura, questo processo dovrebbe risultare relativamente indolore. Se invece ci sono dei problemi, occorre ricontrollare ogni modulo sospetto e, se necessario, modificarlo. In conclusione si avrà un programma perfettamente strutturato e rispondente alle finalità preposte.

Ricapitolando, queste sono le regole per scrivere un programma strutturato:
1. Scrivere una descrizione generale del programma.
2. Suddividerla in moduli secondo i livelli necessari.
3. Per ogni modulo, disegnare un diagramma a blocchi, definire le variabili di input e output e gli altri effetti, quali il tipo di visualizzazione.
4. Scrivere il programma usando, per ogni modulo, le strutture già viste nella parte 1.
5. Provare ogni modulo fornendo l'input e controllando i risultati.
6. Combinare insieme tutti i moduli e provare quindi il programma intero: dovrebbe andare!

I vantaggi di tutta questa fatica sono presto detti: leggibilità, verificabilità, sostituibilità e affidabilità.

I programmi strutturati sono più facili da capire per noi e, per gli altri, più facili da correggere e modificare. Più facili anche da adattare per l'esecuzione su altri computer.

Se queste sono le nostre finalità, allora è il caso di adottare tali tecniche.

COME FUNZIONA IL "BUBBLE SORT"

Per illustrare la costruzione di un modulo, la prova e la connessione cl programma principale, si è usato un programma di ordinamento *"bubble sort"*, una routine presente in vari tipi di programma. Essa ordina alfabeticamente le parole e, in ordine numerico, i numeri. Per quest'ultimo scopo si cambino le variabili stringa da Z$ a Z e le matrici di stringa da A$ ( ) a A( ).

Il computer scorre la lista confrontando gli elementi una coppia per volta. Se questi sono nell'ordine corretto passa oltre, altrimenti li scambia. Il processo si ripete più volte, finché tutti gli elementi sono in ordine.

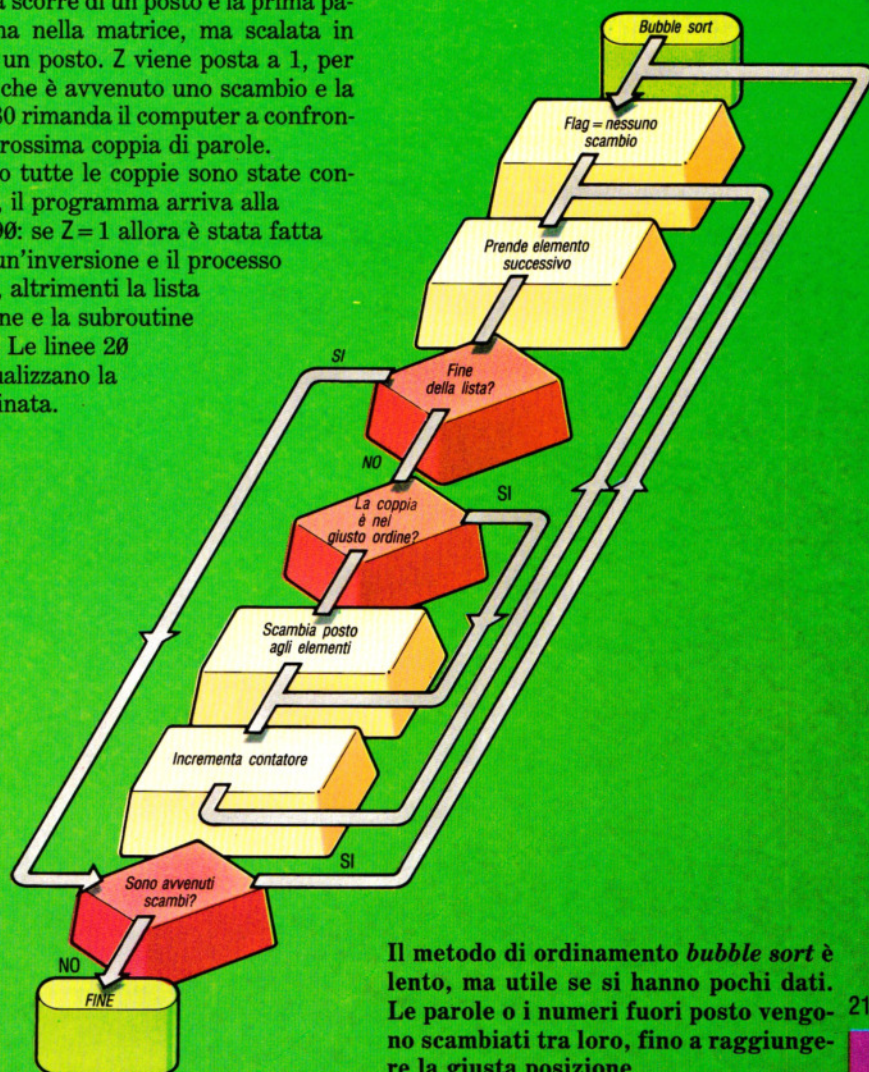
Per vedere il funzionamento del programma nei particolari, lo si confronti con il diagramma a blocchi. La prima parte del programma (non sul diagramma) si informa sul numero di elementi della lista, N e dimensiona la matrice A$ per un numero N di elementi. Le linee 12 e 14 fanno immettere le parole inserendole nella matrice e la linea 16 chiede quali sono da ordinare: se si vuole ordinare l'intera lista, si scriva 1, poi una virgola, poi il numero N. La subroutine è richiamata alla linea 18.

La routine azzera Z, che viene usata come "indicatore" *flag*, in cui registrare se sono avvenuti scambi. La linea 1020 crea un ciclo per scorrere la lista. I valori provvedono a che ogni coppia sia confrontata una volta.

La linea 1030 confronta le prime due parole e, qualora le trovi in ordine, passa alla coppia successiva, saltando la routine di scambio.

Le linee 1040-1070 vengono eseguite soltanto se le parole non erano in ordine corretto. La prima parola viene collocata nella variabile temporanea Z$. La seconda parola scorre di un posto e la prima parola torna nella matrice, ma scalata in basso di un posto. Z viene posta a 1, per indicare che è avvenuto uno scambio e la linea 1080 rimanda il computer a confrontare la prossima coppia di parole.

Quando tutte le coppie sono state confrontate, il programma arriva alla linea 1090: se Z=1 allora è stata fatta almeno un'inversione e il processo si ripete, altrimenti la lista è in ordine e la subroutine termina. Le linee 20 e 22 visualizzano la lista ordinata.

Bubble sort
Flag = nessuno scambio
Prende elemento successivo
Fine della lista?
SI
NO
La coppia è nel giusto ordine?
SI
Scambia posto agli elementi
Incrementa contatore
Sono avvenuti scambi?
SI
NO
FINE

Il metodo di ordinamento *bubble sort* è lento, ma utile se si hanno pochi dati. Le parole o i numeri fuori posto vengono scambiati tra loro, fino a raggiungere la giusta posizione

I CONTROLLI JOYSTICK

Il joystick è uno dei sistemi di controllo alternativo più economici e versatili, non solo per aumentare il divertimento nei giochi, ma anche per usi più seri

Benché gli home computer abbiano raggiunto un livello molto sofisticato, non sono ancora capaci di decidere per conto proprio a quale attività dedicarsi. Questo compito spetterà, ancora per molto tempo, all'operatore. Del resto, l'arte di comunicare col computer (ossia la formulazione di programmi adatti a tal scopo) è uno degli aspetti più avvincenti e al tempo stesso istruttivi dell'intera faccenda.

Nonostante il notevole impegno dei progettisti di hardware nel creare alternative, il principale mezzo di comunicazione col computer, che risale agli albori dell'informatica, è pur sempre la tastiera. Esistono molte ragioni per considerare la tastiera ben lontana dal mezzo ideale, non ultimo il fatto che si tratta di un sistema lento e laborioso, che in aggiunta costringe l'operatore a imparare la dattilografia, parallelamente alle materie pertinenti all'uso del computer.

Per il momento, se vogliamo scrivere programmi, non esiste alcuna valida alternativa alla classica tastiera. Tuttavia essa non serve solo a questo scopo: abbiamo già visto come redigere progrmmi che richiedano l'immissione di dati da parte dell'operatore (vedere alle pagine 129-135). Far sì che il computer reagisca ai dati immessi è una caratteristica di tutti i programmi, siano essi destinati ad applicazioni commerciali o semplici giochi e, ancora una volta, ciò avviene mediante la tastiera.

In questo caso, però esistono due tipi di immissioni. Il primo prevede l'input di dati alfabetici o numerici (ad esempio il nome e la data di nascita di una persona). Il secondo tipo di immissione invece è di tipo analogico: la pressione di un tasto non produce la visualizzazione di un carattere, ma viene bensì sfruttata per funzioni totalmente diverse, quali ad esempio lo spostamento a destra o a sinistra del cursore sullo schermo, in risposta alla pressione dei tasti X o Z.
Abbiamo visto come ottenere queste operazioni alle pagine 54 e 129.

Il primo genere di immissione è decisamente legato all'uso della tastiera, poiché deve consentire la digitazione di un qualsiasi carattere alfabetico o numerico. ma il secondo genere non è costretto a una simile limitazione. Anzi, è decisamente contrario alla logica (e a un facile uso del programma) aspettarsi che l'operatore si annodi le dita, mentre tenta di trovare la P, che muove un'astronave, mentre è occupato a premere la F, il tasto di 'fuoco'.

Fortunatamente, per questo tipo di applicazioni esiste un'alternativa: i *joystick*. Tali dispositivi sono economici e aprono nuovi orizzonti nel campo della comunicazione e del controllo di computer, pur essendo spesso associati unicamente e ingiustamente a un impiego ricreativo.

È vero che i joystick vengono oggi impiegati soprattutto nei giochi, ma le applicazioni possono estendersi anche nel campo professionale.

Per esempio, in un buon programma di grafica è molto più comodo adoperare un joystick (o un simile controllo) per guidare il cursore che disegna sullo schermo (un programma di questo tipo, ma per tastiera, lo si trova a pagina 132) o per selezionare i colori da una "tavolozza", visualizzata sullo schermo.

Addirittura, può essere impiegato un joystick anche per l'immissione di certi dati alfabetici o numerici, purché il programmatore sia capace di impostare il programma in modo opportuno. Un esempio di ciò proviene ancora dal settore dei giochi: il nome del giocatore viene composto (e quindi immesso) selezionando le lettere, tra quelle dell'alfabeto, combinando l'azione del joystick con la pressione del tasto di 'fuoco'. Criteri simili si potrebbero adottare nell'immissioni di nomi, indirizzi e date, oppure per selezionare le opzioni di un menu.

I joystick, quindi, sono degli accessori potenzialmente molto utili in moltissime applicazioni, sostituendo efficacemente la tastiera. Vedremo più avanti nel corso come adattare i propri programmi per avvalersi dei joystick: per adesso, diamo uno sguardo a cosa offre il mercato nel settore hardware.

I TIPI DI JOYSTICK

Non tutti i joystick sono adatti a ogni tipo di computer e occorre accertarsi di acquistarne un modello adatto al proprio computer. Ma c'è un'ulteriore considerazione da fare: nell'acquisto del software, è bene assicurarsi che sia previsto l'uso del particolare joystick in nostro possesso. Con queste premesse, passiamo a esaminare i

vari modelli di joystick.

Il tipo più semplice di joystick è una scatola con una piccola cloche da muovere a destra e a sinistra, in altro, in basso e in diagonale, oltre a un tasto di 'fuoco', che può essere usato anche per vari altri scopi (come per esempio far saltare un personaggio, scegliere una lettera, lanciare una bomba). A seconda del prezzo, esiste un'ampia gamma di varianti all'apparecchio di base.

Alcuni joystick hanno più tasti fuoco, sia per facilitare il gioco, sia per semplificare l'uso da parte dei giocatori mancini. Con appositi programmi, si possono attribuire ad essi diverse funzioni: per esempio, un programma per schedare e uno per redigere.

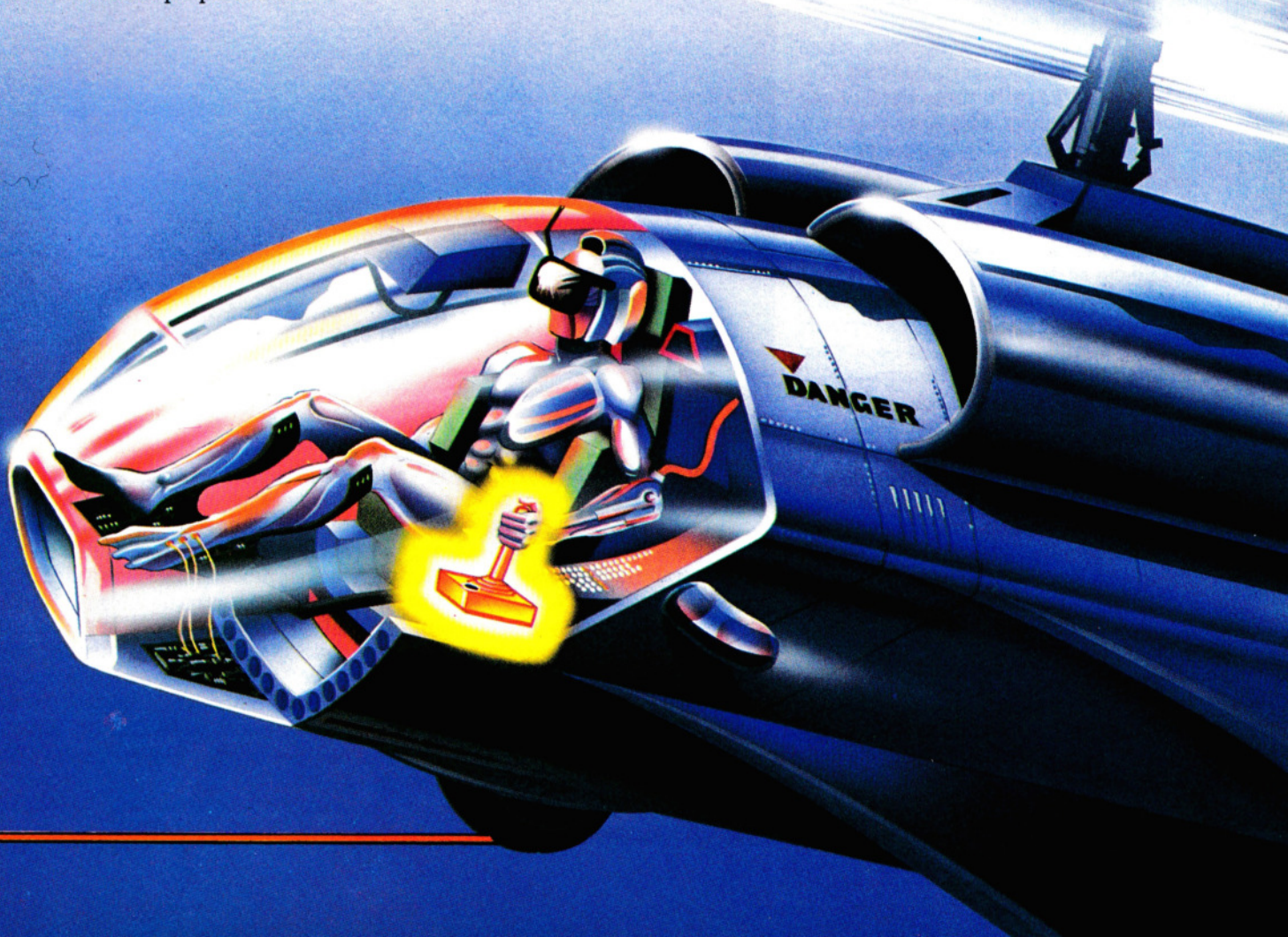
Alcuni tipi sono modellati nella forma della mano, per una presa più salda e più comoda e un controllo più preciso. Esistono modelli a forma di manico di pistola, con tanto di grilletto.

Alcuni dei più recenti modelli non assomigliano nemmeno ai joystick tradizionali. Per esempio, ce ne sono certi in cui il movimento della mano è rilevato mediante interruttori al mercurio, sensibili anche alla minima inclinazione. Una simile sensibilità è anche eccessiva, per certi giochi non scritti appositamente.

Esistono anche tavolette sensibili al tocco, che analizzano la pressione delle dita per rilevare la direzione.

Di un tipo ancora diverso è il *tracker ball*, concepito originariamente per applicazioni aeronautiche, ma ben presto adottato nelle sale giochi e ora anche negli home computer.

Il controllo avviene per mezzo di una sfera, che sporge dalla superficie del contenitore, manovrabile in tutte le direzioni col semplice sfioramento delle dita.

Questi dispositivi di controllo sono anche adatti a usi professionali. Un modello ormai diffuso assieme a recenti personal computer è il "mouse" (o "topo"), una sorta di *tracker ball* rovesciato: il topo funziona facendo rotolare il dispositivo su una superficie piana. Gli spostamenti si

traducono in movimenti del cursore sullo schermo, sostituendo egregiamente la funzione dei tasti cursore. Il posizionamento del cursore, che avviene con molta rapidità, può servire per variare dati o selezionare le voci di un menu.

Le selezioni avvengono tramite i tasti situati sul contenitore e le applicazioni più comuni sono nel disegno su schermo o nei giochi.

Per disegnare, si usano anche tavolette grafiche in cui appositi sensori, nella speciale penna o nello stesso ripiano, consentono di disegnare "a mano libera" o seguendo precisi tracciati. Con le penne ottiche, tale processo avviene direttamente sullo schermo e ne esistono anche versioni per gli home computer.

IL FUNZIONAMENTO

Anche se a prima vista le differenze sembrano enormi, il funzionamento di tutti questi dispositivi è pressoché identico: il movimento è tradotto in una sequenza variabile di segnali elettrici, interpretati dal computer. Come tutti i dispositivi elettronici, i joystick possono essere sia digitali che analogici. Nel tipo digitale il circuito è composto da "interruttori" elettronici che, aperti o chiusi, formano una combinazione binaria unica per ogni direzione prescelta. I tipi analogici contengono invece più potenziometri (resistenze variabili): il movimento comporta una variazione nella quantità di corrente che li attraversa, generando un segnale analogico.

In genere è facile riconoscere un tipo dall'altro: in quello analogico, il collegamento coi due potenziometri è meccanico e la cloche tende a rimanere nella posizione in cui la si lascia. Al contrario, i tipi digitali hano spesso una molla di richiamo, che fa tornare la cloche in posizione centrale. Tuttavia, questa non è una prova decisiva, poiché talvolta anche i primi sono dotati di una molla di richiamo.

Tra i due tipi, inoltre, c'è una sensibile differenza nella *risposta*: quelli del secondo tipo sono più rigidi e si muovono per piccoli tratti. Ciò aggiunge un elemento di realismo (maggiore è la resistenza offerta, maggiore è lo sforzo), ma può ben presto stancare.

Comunque, ciò che influisce maggiormente sull'effetto finale è il software: un programma ben fatto rende anche il joystick più rispondente al controllo. Ad esempio, è il programma che determina se un piccolo spostamento della cloche deve provocare sullo schermo un movimento lento e impercettibile oppure veloce e continuo.

Le tavolette a pressione consistono di due sottili strati di plastica, posti a piccola distanza l'uno dall'altro, sui quali sono posate sottili griglie di resistenze o collegamenti conduttivi, orientati ad angolo retto. Toccando la tavoletta in un qualsiasi punto, si genera un voltaggio preciso analizzato dal computer. Anche alcune tavolette grafiche funzionano in modo analogo, ma su scala più grande. La difficoltà consiste nel produrre tavolette uniformemente sensibili. Purtroppo, se la superficie non è perfettamente pulita, le dita non scorrono bene, rendendo il movimento imperfetto.

I *tracker ball* hanno un funzionamento molto scorrevole, poiché la sfera non è direttamente collegata al sistema sensibile, ma è sostenuta da due rulli, liberi di ruotare in direzione perpendicolare l'una a l'altra.

In qualsiasi direzione la pallina giri, fa ruotare uno o due di questi rulli, collegati ai potenziometri o al sensore digitale. Quest'ultimo è un sistema composto da un diodo LED, da un fototransistor e infine da un disco rotante, opportunamente forato.

Qualsiasi sia il sistema, spetta sempre

A volte i joystick possono causare fastidiose inceppature o problemi di caricamento dei programmi, per i quali si dà troppo spesso la colpa al programma o al registratore. Si verifichi che il joystick sia *saldamente* connesso, prima di caricare il nastro e, sullo Spectrum, si controllino i collegamenti sia dell'interfaccia che delle prese del joystick. Ancora sullo Spectrum, si controlli la compatibilità con il joystick che si sta usando (vedi testo).

al computer tradurre i segnali elettrici in termini di movimento sullo schermo.

COLLEGAMENTO CON IL COMPUTER

A pagina 132 si è visto come scrivere un programma per controllare certe funzioni con la pressione di tasti, usando i comandi GET$ e INKEY$ del BASIC per individuare quale tasto venga premuto.

Fatto ciò, il programma esegue l'operazione associata, ad esempio muovere una base missilistica a destra, se il tasto era quello della X.

In sostanza, l'operazione del joystick è molto simile, sebbene la tastiera sia parte integrante del computer e il joystick no. Occorre quindi collegarlo con il computer mediante la porta adatta o una serie di connessioni, tramite le quali il computer può comunicare con l'esterno.

Il computer va poi programmato allo scopo di esaminare quei terminali alla ricerca del segnale che indica un movimento del joystick.

Ciò è relativamente semplice e in una successiva lezione vedremo come scrivere un programma adatto a trasferire completamente il controllo al joystick. Già fin da ora si capisce che il maggior problema è la compatibilità.

Primo, il joystick deve potersi connettere con la porta del computer, direttamente o tramite un'interfaccia.

Secondo, il programmatore deve conoscere i segnali inviati dal joystick, altrimenti è impossibile farli interpretare dal programma.

Fortunatamente, questa operazione è facilitata dal fatto che i vari costruttori seguono alcuni standard e per ogni micro c'è una precisa serie di regole. Lo standard più comune, ormai universale, è quello del joystick Atari: sono disponibili molti joystick e molti computer aderenti a questo standard.

Il Commodore 64, il Vic 2Ø e lo Spectrum operano con controlli di tipo bilanciato al centro (digitale). Il BBC, il Dragon e il Tandy usano tipi non bilanciati al centro (a potenziometri), anche se sono disponibili manopole di tipo bilanciato al centro con potenziometro. All'interno di questa gamma, esistono altre differenziazioni, ma la più importante riguarda l'interfacciamento.

INTERFACCIAMENTO

Come ogni altra periferica per computer, i joystick devono interfacciarsi con la macchina.

L'interfaccia può essere già incorporata nella configurazione del computer, oppure essere un dispositivo a sé . Se si può collegare un joystick direttamente, in genere significa che l'interfaccia è già presente nel computer ed è probabile che segua lo standard Atari.

Alternativamente, l'apparecchio può essere dotato di una o più porte analogiche, alle quali collegare i potenziometri, ma è una soluzione meno diffusa, poiché per ogni joystick usato vanno incorporati due convertitori analogico/digitali.

La scelta di un joystick per uno Spectrum è più difficile dato che la gamma in commercio è la più estesa di tutte. I joystick per lo Spectrum sono talmente numerosi che alcuni programmi hanno addirittura un menu per adattare automaticamente il programma al tipo di joystick impiegato sul computer.

In effetti, la differenza non sta tanto nei joystick in sé, quanto nelle interfacce, dal momento che la Sinclair non fornisce un'interfaccia joystick assieme a questi computer.

Prima di usare il joystick, occorre collegare un'apposita interfaccia al connettore che è situato posteriormente all'apparecchio.

L'acquisto dell'interfaccia, per i possessori di Spectrum, significa purtroppo raddoppiare la spesa.

Di solito, l'interfaccia è alloggiata in un contenitore a parte, innestato sul retro della macchina e a diretto contatto con i terminali del computrer. Esiste tuttavia un tipo di joystick con interfaccia incorporata, ma sembra essere meno diffusa tra gli utenti.

Alcuni joystick per lo Spectrum seguono lo standard Atari, ma altri utilizzano terminali diversi nella porta utente. Il software esamina particolari terminali, per leggere i segnali inviati dall'interfaccia del joystick: usando modelli non adatti, non è detto che i segnali compaiano al posto giusto.

Chi, tra i possessori di Spectrum, abbia già acquistato una nutrita collezione di programmi, può trovarsi ad averne alcuni non adatti a un particolare standard del joystick. È consigliabile, prima di acquistare il joystick, selezionarlo in base alle necessità del software già posseduto o che si intende comprare.

In genere ci si trova di fronte a quattro possibili scelte.

La prima è rinunciare a usare il joystick con alcuni dei programmi: una scelta drastica, che evita però una quantità di spese imprevedibili. In alternativa, esiste un joystick "rudimentale", ossia una cloche meccanica, che si sovrappone ai tasti e opera direttamente su questi.

La seconda alternativa è l'acquisto di due o più interfacce, ma si tratta di una scelta costosa.

La terza alternativa è l'acquisto di un'interfaccia programmabile, molto più costosa del tipo comune. Però l'investimento vale, perché un simile accessorio rende il joystick compatibile con qualsiasi programma, anche con giochi che non ne prevedono l'uso. Questi modelli permettono di definire, attraverso le porte usate dalla tastiera, a quali tasti associare il joystick. In questo modo l'utente 'fa credere' allo Spectrum che i movimenti del joystick siano pressioni di tasti. L'unico svantaggio dell'interfaccia programmabile è che va riprogrammata ogni volta che si usa con un diverso standard.

La quarta scelta è l'acquisto di "cassette di compatibilità", che permettono di usare i diversi standard del joystick. Costano circa quanto un gioco e in pratica permettono di usare il joystick con qualsiasi programma in commercio. Lo svantaggio maggiore è che occorre caricare due programmi ogni volta che si vuole usare un gioco.

L'interfaccia specifica del Sinclair, l'Interfaccia 2, permette di collegare due joystick per volta, opponendo così due giocatori, sempre che il software preveda questa opzione. Permette anche di usare le cartucce ROM, recentemente commercializzate, ma purtroppo esiste ancora uno scarso software compatibile con questa interfaccia.

Perché esistono tanti sistemi alternativi di controllo

Progettisti e costruttori hanno messo a punto un gran numero di alternative al controllo da tastiera, ossia sistemi che permettono all'utente di comunicare in qualche modo le istruzioni al computer. Molti di questi dispositivi non sono ancora disponibili su larga scala e pertanto non si ha un campione valido dei pregi di un progetto rispetto a un altro.

Inoltre, la maggior parte di questi sistemi vengono sviluppati per applicazioni specifiche o per necessità particolari, come la tavoletta grafica per le aziende di grafica computerizzata, o il joystick e il "topo" per alcuni word processor, per esempio. L'uso dei più disparati dispositivi, poi, si è andato spesso diversificando in aree in cui proliferano modelli sconosciuti.

In genere, per usi non professionali, il joystick garantisce ancor il miglior rapporto costi/prestazioni, benché saranno presto reperibili sistemi più economici ed efficaci.

Lo ZX81 non è l'apparecchio ideale per i giochi, ma, come lo Spectrum, può essere interfacciato con un joystick, mediante operazioni per lo più simili a quelle ora descritte. La spesa di un simile sistema è però molto alta.

Il Commodore 64 e il Vic 20 hanno entrambi un paio di prese per joystick digitali. Ciò permette di far gareggiare due giocatori, se il programma prevede questa opzione. Dato che le due prese sono connesse a un'interfaccia che segue lo standard Atari, i possessori di questi apparecchi hanno una possibilità di scelta molto ampia tra la gamma di joystick delle varie marche.

I computer BBC usano joystick a potenziometro, direttamente collegati alla porta analogica. il Dragon e il Tandy sono simili, ma i joystick che vanno bene per questi apparecchi non sono adatti per i computer BBC a causa del diverso tipo di connettore impiegato. Non c'è molta scelta per i possessori di Dragon, benché la Dragon Data produca i propri joystick. Chi possiede l'Electron ha la possibilità di acquistare interfacce compatibili con lo standard Atari.

L'ACQUISTO DI UN JOYSTICK

La prima cosa, e la più importante, è assicurarsi che qualsiasi modello si compri sia compatibile con il proprio computer e anche con il proprio software.

Poi si valuti quanto si vuole spendere e si tenga a mente che un'interfaccia può costare quanto e più del joystick. Il tipo di joystick più semplice costa quanto un gioco, mentre i tipi più elaborati possono costare anche due o tre volte tanto.

All'estremo opposto (con sei volte il costo del tipo più economico di joystick), ci si può permettere il più economico dei *tracker ball*, mentre il più costoso costa quanto lo stesso computer.

Se è possibile, si chieda al negoziante una dimostrazione sull'impiego del joystick. Si tenga presente anche che un appassionato di giochi sottopone un joystick a molte ore di uso logorante.

INDICE CUMULATIVO DEI FASCICOLI

NEL PROSSIMO NUMERO

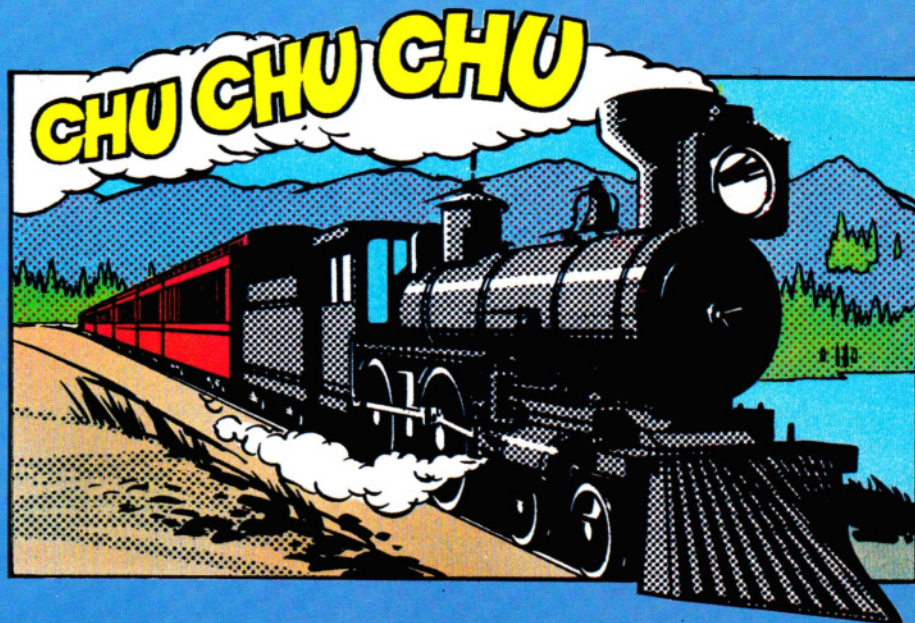
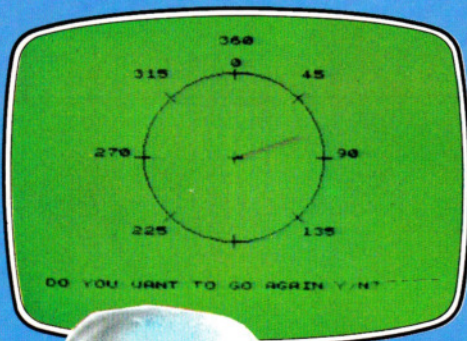
☐ *Il MICROPROCESSORE controlla tutte le funzioni del computer: come funziona?*

☐ *Movimentiamo lo schermo del Dragon e del Tandy con alcuni UDG A COLORI.*

☐ *Vediamo come SIN e COS possano migliorare i nostri disegni al computer.*

☐ *Impariamo come le istruzioni PEEK e POKE servano per modificare la memoria del computer.*

☐ *Aggiungiamo EFFETTI AUDIO ai nostri programmi di gioco.*

☐ *Tutto sulle STAMPANTI e come scegliere la più adatta.*

CHIEDETE INPUT AL VOSTRO EDICOLANTE